2000

The distributed server network design problem

Christopher Howard Stacey
University of Wollongong

THE DISTRIBUTED SERVER NETWORK DESIGN PROBLEM

A thesis submitted in fulfilment of the requirements
for the award of the degree

DOCTOR OF PHILOSOPHY

from

UNIVERSITY OF WOLLONGONG

by

Christopher Howard Edmett Stacey
BSc (Hons I)

Department of Electrical, Computer & Telecommunications Engineering
2000

To God be the Glory.

Abstract

This thesis considers the design of a *Distributed Server Network* (*DSN*). DSN's are required to support the growing number of services where users access resources via a network. Example network services include: the World Wide Web (WWW), distributed database services such as large corporate databases, banking systems, videoconferencing, network games, the Internet's Domain Name System (DNS), Federations of Traders in Open Distributed Processing (ODP) environment, and the X.500 directory service.

A DSN comprises an access network connecting clients to servers, and a backbone network interconnecting servers. We call the design of a DSN the *Distributed Server Network Design Problem* (*DSNDP*). We examine two closely related problems in the design of DSN's. The first is the design of an entirely new network, given the location of clients, the volume of traffic generated (and required) by each and the possible locations of servers. Here the task is to select: (i) the number, location and capacity of backbone nodes (servers), (ii) the assignment of client nodes to servers, and (iii), the link topology and capacity of both the access and backbone portions of the network. The second form of the problem is the same as the first, except that a link topology is already in place. Here we design a virtual network of data paths and determine the capacity required by them.

Three heuristic design procedures for the DSNDP are developed and compared in this dissertation. One procedure is based on a node clustering approach, while the other two are based on a greedy search of the solution space using either ADD/DROP or ADD-*k* heuristics. In addition we develop a procedure to provide a lower bound on the cost of a optimal DSN design using a *continuous branch-and-bound* algorithm involving both linear and *Sequential Quadratic Programming* (*SQP*).

A key feature of all three DSN design procedures is that they decompose the problem into two parts: (i) the location of servers and assignment of client nodes to them, and (ii) the design of the link topologies connecting clients to servers and

servers to one another. The design of the link topologies, once servers are located and clients assigned to them, is a classic network link *Topology, Capacity and Flow Assignment* (*TCFA*) problem [Klei76] [Gerl77]. We develop a *Concave Link Elimination* (*CLE*) algorithm to solve the TCFA problem in the presence of strongly concave link cost functions. All three of our DSN design procedures use the CLE procedure to design the link topology of the DSN's they produce. A lower bounding procedure is also developed for the Concave TCFA problem using continuous branch-and-bound and SQP. The lower bounding procedure allows us to assess the performance of not only our CLE procedure, but also other algorithms suitable for solving Concave TCFA problems, such as Kleinrock and Gerla's *Concave Branch Elimination* (*CBE*) procedure [Klei76] [Gerl77], and Gersht and Weihmayer's *greedy link elimination* procedure [Gers90].

An extensive performance comparison of our CLE procedure with the two TCFA procedures from the literature is presented. Our results show that the CLE procedure is able to produces network designs whose cost is within 1% (on average) of those produced by Gersht's procedure, in significantly less time (the CLE procedure is almost two orders of magnitude faster than Gersht's procedure in designing 20 node networks). When compared to the CBE procedure the cost of the designs produced by the CLE procedure are 45% (on average) cheaper, and again take less time to produce.

An extensive performance comparison of the three DSN design procedures (which all employ the CLE procedure) is also presented. Our results show that while the cost of the solutions produced by the three procedures are very similar (there is less than 2% difference between them in most cases), there is a significant difference in the complexity and hence execution time of the procedures.

Our comparison of the heuristic design procedures leads us to conclude that good quality solutions to both the Concave TCFA problem and DSNDP can be obtained, in reasonable time, using the design procedures described in this dissertation. The Concave TCFA problem can be solved using the CLE procedure. Of the three DSN design procedures, the ADD/DROP procedure represents the best trade-off between solution quality and execution time.

Acknowledgments

I would like to acknowledge the efforts of my supervisors Tony Eyers, and Gary Anido. I am quite sure that without Tony's mostly patient encouragement I would never have completed this task. I thank Gary for teaching me, a sometimes reluctant student, to never simply accept things, but to challenge everything. I'd also, like to thank Hugh Bradlow for helping to start me on the road to a PhD, I learnt a great deal from him in a short time and wish I'd had longer.

During my PhD I have been privileged to be supported by funding from Telstra Research Labs. The financial assistance has made my life a great deal easier, and allowed to concentrate on my research full time. In addition, the opportunity to bounce ideas off the researchers at TRL has been instrumental in guiding the direction of my research.

For the first three years of this project I worked, and played, alongside an ever fascinating mixture of people in Switched Networks Research Centre (SNRC) lab. I'd like to thank all the SNRC'ers for the fun times, interesting discussions and good meals had together. Over the last three years I've swapped a great group of friends in Australia for an equally valuable collection at Motorola in Swindon, UK. My thanks also to them

To my parents, my thanks for always believing in me, and giving me the opportunity to fulfil my dreams. My only regret is that my father has not lived to see this task through to completion. In many respects I've done this PhD more for him than me. I wish he was still with us to see the final article.

My deepest thanks and love to my wife. Not only has she tolerated my ups and downs, supported and encouraged me at every step along the way, she has done so while completing her own PhD.

Finally, to God, who knew me before I was born, who has never forsaken me, nor left me, thank you.

Author's Notes

The spelling used in this dissertation conforms to an "English" rather than "American" convention. For example, the word optimisation is spelt with an 's' rather than a 'z'. The only conscious exceptions to this rule are within quoted material, or titles.

Where fractional numbers are used in the text the "English/American/Australian" formatting convention is used. That is, a dot '.' as a decimal point, and a comma ',' as a separator between thousands, millions, and so on. For example, the number one thousand, point six is represented as 1,000.6.

Table of Contents

Abstract ii

Declaration iv

Acknowledgments v

Author's Notes vi

Table of Contents vii

List of Figures x

List of Tables xiii

List of Abbreviations xiv

1. Introduction 1

1.1 Distributed Server Network Design 1
1.2 Dissertation Outline 4
1.3 Contributions of this Dissertation 6
1.4 Publications Based on this Dissertation 8
1.4.1 Published 8
1.4.2 Accepted for Publication 9

2. A Survey of Existing Network Design Techniques 10

2.1 Introduction 10
2.2 Facility Location Problems 11
2.2.1 Branch and Bound Based Approaches 13
2.2.2 Lagrangian Relaxation Based Approaches 14
2.2.3 Greedy Search Based Approaches 17
2.2.4 Conclusions 18
2.3 p-Median Problems 19
2.3.1 Solution Approaches 20
2.3.2 Conclusions 23
2.4 Hub Location Problems 23
2.4.1 Exhaustive Search Based Approaches 25
2.4.2 Node Clustering and Exchange Based Approaches 26
2.4.3 Conclusions 29
2.5 Centralised Network Design 30
2.5.1 Greedy Search Based Approaches 32
2.5.2 Conclusions 36
2.6 Backbone Network Design or the TCFA Problem 37
2.6.1 Decomposition Based Approaches 38
2.6.2 Mathematical Programming and Lagrangian Relaxation Based Approaches 39
2.6.3 Greedy Link Elimination Based Approaches 40
2.6.4 Extensions of the TCFA Problem 44

2.6.5 Conclusions 47
2.7 Conclusions 48

3. Problem Formulation and Solution Approaches 52

3.1 Introduction 52
3.2 Mathematical Programming Formulation 52
3.2.1 Assumptions 53
3.2.2 Input Variables 54
3.2.3 Decision (or Output) Variables 54
3.2.4 Network Component Cost Functions 55
3.2.5 Mixed Integer Non-Linear Programming Formulation 56
3.3 Combined Network Design Problems 59
3.3.1 Facility Location and Centralised Network Design 59
3.3.2 Hub Location and Backbone Network Design 64
3.3.3 Distributed Database and Network Topology Design 72
3.4 Problem Decomposition & Solution Approaches 73
3.4.1 Global Search Techniques 75
3.4.2 Local or Directed Search Techniques 76
3.5 Conclusions 77
3.6 Deficiencies & Developments 78
3.6.1 Deficiencies in Previous Work 78
3.6.2 Developments in this Dissertation 80

4. The Concave TCFA Problem 83

4.1 Introduction 83
4.2 Mathematical Programming Formulation 85
4.3 Reformulating to Find a Lower Bound 88
4.4 Continuous Branch-and-Bound 91
4.4.1 Continuous Branch-and-Bound in Operation 93
4.4.2 Forming Convex Relaxations 95
4.4.3 Improving the Upper Bound 96
4.4.4 Time and Memory Requirements 97
4.5 Concave Branch Elimination (CBE) Procedure 98
4.6 Concave Link Elimination (CLE) Procedure 100
4.7 CLE vs. Gersht vs. CBE vs. Lower Bound 105
4.7.1 Random Network Design Problem Generation 105
4.7.2 Solution Cost and Execution Time Results 107
4.8 Conclusions 111

5. Distributed Server Network Design Procedures 112

5.1 Introduction 112
5.2 Node Clustering Procedure 113
5.2.1 General Operation of Clustering Algorithms 113
5.2.2 Node Clustering in the Network Design Literature 115
5.2.3 Proposed Node Clustering Procedure 117
5.2.4 Clustering Algorithm Variations 122
5.3 Greedy Local Search Procedures 124

5.3.1 ADD/DROP Heuristic 125
5.3.2 ADD-k Heuristic 126
5.4 Complexity and Execution Time Analysis 128
5.5 Heuristic Quality Assurance 131
5.5.1 Formulation of a Lower Bounding Problem 131
5.5.2 Solving the Reduced Lower Bounding Problem via Continuous Branch & Bound 133
5.6 Conclusions 136

6. Design Methodology & Performance Comparison 138

6.1 Introduction 138
6.2 A Distributed Server Network Design Methodology 138
6.2.1 Component Cost Function Parameters 139
6.2.2 Client Demand Topology and Volume 140
6.2.3 Design Constraints 141
6.2.4 Further DSN Variations 141
6.3 A Specific Network Example 145
6.4 Random Network Results 149
6.4.1 Performance Comparison of the Clustering Algorithm Variations 150
6.4.2 Clustering vs. Greedy Algorithms 156
6.5 Clustering vs. Greedy Procedure Comparison Conclusions 162

7. Conclusions 165

7.1 Introduction 165
7.2 The Concave TCFA Problem 166
7.3 The DSN Location-Allocation Problem 168
7.4 Aiding the DSN Researcher and Designer 170
7.5 Areas for Further Work 172
7.5.1 DSN Stability Analysis 173
7.5.2 DSN Expansion 173
7.5.3 Multiperiod DSN Design 174

References 175

Appendix A. Additional Results from a Performance Comparison of Variations of the Node Clustering DSN Design Algorithm 186

Appendix B. Additional Results from a Performance Comparison of the Node Clustering, ADD/DROP, and ADD-k DSN Design Procedures 190

List of Figures

Figure	Title	Page
1.1	An example DSN design.	1
2.1	An example solution of a facility location problem.	12
2.2	An example solution of a p-median problem, on a tree network.	20
2.3	An example hub based network design.	24
2.4	An example centralised network design.	32
2.5	An example backbone network design.	38
2.6	An illustration of trombone routing in operation.	41
2.7	An illustration of how the incremental increase in network cost due to the elimination of a link can increase when concave link cost functions are employed.	43
2.8	An illustration that tree networks are cheaper than meshed networks when the link cost functions are concave with respect to capacity and linear with respect to distance.	46
3.1	Star-backbone path topology for centralised network design from [Pirk92].	62
3.2	Representation of client node assignments as n digit base k numbers.	74
4.1	Continuous branch-and-bound in operation.	94
4.2	CBE vs. Gersht vs. CLE: Solution cost comparison.	108
4.3	CBE vs. Gersht vs. CLE vs. Bounds: Solution cost comparison.	109
4.4	CBE vs. Gersht vs. CLE: Execution time comparison.	110
5.1	Execution time comparison for randomly generated networks without preexisting link topologies.	129
5.2	Memory and execution time requirements of the continuous branch-and-bound procedure used to solve the DSN lower bounding problem vs. the bounding aggressiveness parameter, .	135
5.3	Execution time requirements of, and relative cost of solutions produced by, the continuous branch-and-bound procedure used to solve the DSN lower bounding problem vs. a range of values of the bounding aggressiveness parameter, .	136
6.1	A specific 22 node network example from [Saha95], with a preexisting link topology generated by the CLE algorithm.	146
6.2	Design of 22 node network, with existing link topology, produced by both fast and slow clustering DSN design procedures.	148
6.3	Design of 22 node network, with existing link topology, produced by both the ADD/DROP and ADD-k DSN design procedures.	149
6.4	Relative cost of solutions produced by variations of the clustering procedure as the size of the network varies. Network topologies are unconstrained.	151
6.5	Execution time of variations of the clustering procedure as the size of the network	

varies. Network topologies are unconstrained. 151

6.6 Relative cost of solutions produced by variations of the clustering procedure as the relative cost of servers and links varies. Network topologies are unconstrained. 153

6.7 Relative cost of solutions produced by variations of the clustering procedure as the ratio of inter-server to client-server traffic varies. Network topologies are unconstrained. 154

6.8 Relative cost of solutions produced by variations of the clustering procedure as the number of potential server locations varies. Network topologies are unconstrained. 155

6.9 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-k procedures as the size of the network varies. Network topologies are unconstrained. 157

6.10 Relative cost of solutions produced by the clustering, ADD/DROP, ADD-k, and lower bounding procedures as the size of the network varies. Network topologies are unconstrained. 158

6.11 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-k procedures as the relative cost of servers and links varies. Network topologies are unconstrained. 159

6.12 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-k procedures as the ratio of inter-server to client-server traffic varies. Network topologies are unconstrained. 160

6.13 Relative cost of solutions produced by the clustering, ADD/DROP, ADD-k, and lower bounding procedures as the ratio of inter-server to client-server traffic varies. Network topologies are unconstrained. 161

6.14 Relative cost of solutions produced by the clustering, ADD/DROP, and Add-k procedures as the number of potential server locations varies. Network topologies are unconstrained. 162

A.1 Relative cost of solutions produced by variations of the clustering procedure as the size of the network varies. Network topologies are constrained. 187

A.2 Execution time of variations of the clustering procedure as the size of the network varies. Network topologies are constrained. 187

A.3 Relative cost of solutions produced by variations of the clustering procedure as the relative cost of servers and links varies. Network topologies are constrained. 188

A.4 Relative cost of solutions produced by variations of the clustering procedure as the ratio of inter-server to client-server traffic varies. Network topologies are constrained. 189

A.5 Relative cost of solutions produced by variations of the clustering procedure as the number of potential server locations varies. Network topologies are constrained. 189

B.1 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-k procedures as the size of the network varies. Network topologies are constrained. 191

B.2 Relative cost of solutions produced by the clustering, ADD/DROP, ADD-k, and lower bounding procedures as the size of the network varies. Network topologies

are constrained. 191

B.3 Relative cost of solutions produced by the clustering, Add/DROP, and ADD-k procedures as the relative cost of servers and links varies. Network topologies are constrained. 192

B.4 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-k procedures as the ratio of inter-server to client-server traffic varies. Network topologies are constrained. 193

B.5 Relative cost of solutions produced by the clustering, ADD/DROP, ADD-k, and lower bounding procedures as the ratio of inter-server to client-server traffic varies. Network topologies are constrained. 193

B.6 Relative cost of solutions produced by the clustering, ADD/DROP, and Add-k procedures as the number of potential server locations varies. Network topologies are constrained. 194

List of Tables

Table	Title	Page
4.1	Link Capacities and Corresponding Costs	86
4.2	Double power-law function (Equations (4.1) and (4.2)) parameters fitted to cost data.	86
5.1	Node clustering based DSN design procedure variations.	124
6.1	Server Capacities and Corresponding Costs	139
6.2	Double power-law cost function (Equations (3.3), (3.4) and (3.5)) parameters fitted to cost data.	140
6.3	22 node network, with an existing link topology, DSN design procedure results.	147
6.4	22 node network, without an existing link topology, DSN design procedure results.	147

List of Abbreviations

ATM	Asynchronous Transfer Mode
b/s	Bits per second
BER	Bit Error Rate
CA	Capacity Assignment
CBE	Concave Branch Elimination
CFA	Capacity and Flow Assignment
CFLP	Capacitated Facility Location Problem
CLE	Concave Link Elimination
CMST	Capacitated Minimum Spanning Tree
DNS	Domain Name Service
DSN	Distributed Server Network
DSNDP	Distributed Server Network Design Problem
FA	Flow Assignment
FD	Flow Deviation
FLP	Facility Location Problem
FOGA	First Order Greedy Algorithm
FTP	File Transport Protocol
GA	Genetic Algorithm
GoS	Grade of Service
HTTP	Hypertext Transport Protocol
IP	Integer Program
LNSDP	Light Network Server Design Problem
LP	Linear Program
Mb/s	Megabits per second
MCL	Minimum Cost Loop
MCT	Multicentre Capacitated Tree
MILP	Mixed Integer Linear Program
MINLP	Mixed Integer Non-Linear Program
MST	Minimum Spanning Tree
NLP	Non-Linear Program

NNTP	Network News Transport Protocol
PSA	Parallel Savings Algorithm
QoS	Quality of Service
SA	Simulated Annealing
SOGA	Second Order Greedy Algorithm
SQP	Sequential Quadratic Programming
SSCLP	Star-Star Concentrator Location Problem
TCFA	Topology, Capacity and Flow Assignment
UFLP	Uncapacitated Facility Location Problem
VSN	Virtual Service Network

1. Introduction

Great works are performed not by strength, but perseverance.

- Samuel Johnson

1.1 Distributed Server Network Design

In this dissertation, we consider the design of a *Distributed Server Network* (*DSN*). A DSN is a collection of interconnected servers that communicate with one another to support service requests from clients on a network wide basis. Structurally, a DSN comprises an access network connecting clients to servers, and a backbone network interconnecting servers. We refer to the task of designing a DSN as a *Distributed Server Network Design Problem* (*DSNDP*). Any solution of a DSNDP will determine: (i) the number, location and capacity of servers; (ii) the assignment of clients to servers; (iii) the capacity and topology of links; and (iv) the routing of traffic in both the access network connecting clients to servers, and the inter-server network. Figure 1.1 shows an example DSN design

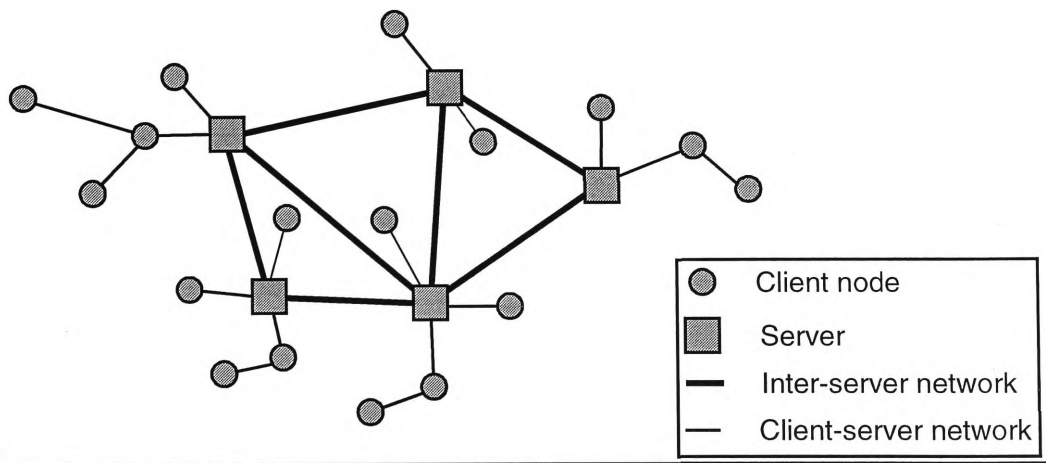


Figure 1.1 An example DSN design.

DSN's are required to support the growing number of services where users access server resources that are distributed about a network. The resources of the servers may be employed to provide a variety of services, such as:

- Real-time, multimedia, multi-user services such as video and/or audio conferencing, or network games. In this instance the servers would act as conferencing bridges, merging and distributing user traffic.
- Distributed database applications - in this context not all servers store the same information. Users send queries to their assigned server, which may have to query other servers to satisfy the user requests. Depending on the service, users may send updates to their server, which may in turn be propagated to other servers in the network. Common examples of this type of service include: the World Wide Web (WWW), the Internet's Domain Name System (DNS), "federations" of Traders in an Open Distributed Processing (ODP) environment, large corporate or banking database systems, X.500 directory services, and Microsoft's Active Directory™ service.
- Duplicated database applications - in this context all databases store the same information. All requests for a given user are handled by the copy of the database (i.e. server) to which they are assigned. Inter-server traffic is generated by database updates. This type of environment would arise when providing a fault tolerant database service.

Rather than focus on any specific application we keep our model of a DSN sufficiently general that by modifying a few input parameters we are able to design a network to support any of the service types described above.

We examine two closely related problems in the design of DSN's. In the first instance we examine the problem posed in a "green fields" environment. That is, the design of an entirely new network from scratch when given only the location of clients, the volume of traffic generated (and required) by each client, and the possible locations of servers. The second form of the problem is that posed by the design of a DSN whose final form is restricted by an existing network topology. In this environment we are effectively designing a virtual network of data paths (and their capacity) to be super-imposed onto an existing network. In either case, the task is to select: (i) the number, location and capacity of backbone nodes (servers), (ii) the

assignment of client nodes to servers, and (iii), the link (or virtual path) topology of both the access and backbone portions of the network.

The DSNDP combines aspects of both location-allocation, and network topology design problems. Determining the number and location of servers is a location problem. Assigning user nodes to servers is an allocation (or assignment) problem. The DSNDP also includes the design of the supporting network topology. Thus the DSNDP can be described as a constrained, single allocation, interacting hub location and network design problem. This type of problem has been examined in the literature in various forms. However, as will be shown, the complete design of a DSN has not been adequately addressed by previous work.

For modelling purposes, a communications network can be represented as a directed graph of nodes connected by edges. Each node represents the location of a group of users. Each node may also be a potential server location. Each edge represents either a physical link between two nodes or in the case of an ATM network, a virtual path supported by an underlying physical infrastructure. The edges in the final solution represent the physical links or virtual paths needed to provide the DSN.

The DSNDP can be stated as follows:

Given:

- the location of client nodes,
- the size of the client population located at each node (and hence the demand that each node will place on its assigned server),
- a set of potential server locations (i.e., those nodes at which we may place servers),
- the level of interaction between servers,
- the cost functions for both links and servers,

subject to:

- link and/or server capacity limits,
- a limit on the average packet queueing delay,
- and optionally, an existing link topology to which the design is constrained.

Determine:

- the number, location and capacity of servers,
- an assignment of client nodes to servers,
- the topology and capacity of network links connecting client to servers and servers to each other,

in order to:

- minimise the total (link and server) cost of the network.

1.2 Dissertation Outline

This dissertation is divided into seven chapters. A broad outline of the dissertation is as follows:

This chapter provides an introduction to the DSNDP, an overview of the dissertation as a whole, a summary of the contributions made in this dissertation, and a list of publications arising from this work.

Due to the complexity of problems such as the DSNDP, much of the previous work on network design has focused on solving specific sub-problems which contribute to the solution of the whole problem. Chapter 2 reviews previous work on specific network design problems which are sub-problems of the DSNDP. Furthermore, the existing solution procedures for these sub-problems are not directly applicable to the DSNDP in their current form.

Recently there has been recognition that complex network design problems, such as the DSNDP, are best solved as a whole rather than decomposing them and solving the sub-problems separately [Gavi92a]. Chapter 3 reviews previous work on problems that combine two or more of the problems described in Chapter 2. We first present the DSNDP as a mathematical program (see Section 3.2), to illustrate how it combines various problems from Chapter 2. Having discussed previous work on similar combined problems in Section 3.3, we discuss possible solution methods for the DSNDP in Section 3.4. We observe that once the location of servers and the assignment of client nodes to them is known, the traffic requirements between all

node pairs is also known. This leads us to search for good solutions by selecting server locations and client node assignments and then solve the network topology design problem.

The evaluation of any given configuration of servers and their associated clients requires the solution of a complex network topology design problem. Our Concave Link Elimination (CLE) procedure for solving the network link Topology, Capacity and Flow Assignment (TCFA) problem in the presence of strongly concave link cost functions is presented in Chapter 4. The performance (in terms of solution cost and execution time) of our CLE procedure is compared with existing procedures. In addition, a procedure to produce a lower bound on the cost of the optimal solution to the problem is developed.

Chapter 5 presents two varieties of directed search procedures for determining the optimal configuration of servers and assigning client nodes to them. Three procedures are developed, one based on a node clustering approaches and two on greedy searches. All three procedures employ the CLE procedure developed in Chapter 4 to design the network topologies. In addition, a procedure to provide a lower bound on the cost of the optimal solution is developed.

A methodology to solve the DSNDP employing the algorithms developed in Chapters 4 and 5, is presented in Chapter 6. This methodology is combined with the lower bounding procedure to compare the performance of the three procedures.

The net result of Chapters 4 through 6 is a collection of procedures and a methodology for the design of near optimal DSN's. The main findings of this thesis, along with suggestions for future research areas are presented in Chapter 7. To summarise, this dissertation presents a number of new techniques, developed by building on various areas of the existing work on network design, that able to provide near optimal solutions to the DSNDP in reasonable time. In addition, lower bounding procedures that allow the quality of DSNDP solutions to be evaluated are also provided. Finally, we outline a design methodology that illustrates how a designer can use all of our design procedures together to rapidly produce prototype DSN designs, which can then be refined. An extensive performance comparison of all of the procedures shows that our procedures significantly out-perform those in the literature. The next section summarises the main contributions of this dissertation.

1.3 Contributions of this Dissertation

1. Use of a continuous double power-law function to model discrete server and link costs. Unlike offset linear or power-law cost functions (commonly used in the literature), the double power-law function captures non-linear economies of scale in component costs and hence is more realistic. In addition the double power-law is smooth, allowing the use of gradient based optimisation techniques. See Section 3.2 and first published in [Stac96a] and [Stac96b].

2. Identification of the two part nature of the DSNDP. Essentially, it consists of both client-server and inter-server traffic flows, related through the subset of servers chosen and assignment of clients to servers. This observation led to the development of heuristic design procedures based on the decomposition of the problem into the location of servers (and assignment of client nodes to them) and the design of the client to server and inter-server portions of the network. See Section 3.4 and first published in [Stac96a] and [Stac96b].

3. Enhancement of a greedy link elimination procedure from [Gers90] leading to the development of a Concave Link Elimination (CLE) procedure to solve Concave TCFA network design problems and thus solve the network topology design portion of the DSNDP. See Section 4.6 and first published in [Stac97b].

4. Development of a lower bounding procedure to obtain a lower bound on the cost of the optimal solution to any given concave TCFA problem using a continuous branch-and-bound algorithm and Sequential Quadratic Programming (SQP). See Sections 4.2, 4.3, 4.4 and first published in [Stac97b]. This provides an absolute measure of the quality of the solutions produced by a Concave TCFA solution procedure.

5. Extensive comparison of the performance of the CBE, CLE and Gersht's solution procedures. Our results show that our CLE procedure is able to produce network designs whose cost is within 1% (on average) of those produced by Gersht's procedure, in significantly less time (the CLE procedure is almost two orders of magnitude faster than Gersht's procedure in designing 20 node networks). When compared to the CBE procedure the cost of the

designs produced by the CLE procedure are 45% (on average) cheaper, and again take less time to produce. See Section 4.7 and first published in [Stac97b].

6. Development of a node clustering heuristic to solve the location-allocation portion of the DSNDP. Furthermore, we show that our heuristic procedures perform significantly better than an existing node clustering network design procedure. An early version of this algorithm appeared in [Stac96c]. The final version is presented in Section 5.2 and [Stac96b].

7. Adaption of two existing forms of greedy search heuristics to solve the location-allocation portion of the DSNDP. The first, a common ADD/DROP greedy search heuristic to solve the DSNDP, is presented in Section 5.3.1 and [Stac96b]. Secondly, an ADD-*k* greedy search heuristic from [Gavi90] is presented in Section 5.3.2 and [Stac96b]. Although, both types of heuristic have appeared in the literature before neither has appeared in a form directly applicable to the DSNDP. Our results in Chapter 6 show that our versions of these heuristics to produce near-optimal solutions to the DSNDP.

8. The development of a lower bounding relaxation of the full DSNDP and its solution using continuous branch-and-bound and SQP algorithms. See Sections 5.5.1 and 5.5.2. This work also appeared in [Stac97a] and [Stac97b]. This provides an absolute measure of the quality of the solutions produced by a DSN design procedure.

9. Development of a methodology to both solve the DSNDP and compare the performance of the design algorithms employed. See Section 6.2 and first published in [Stac96a] and [Stac96b]. Thus this dissertation provides a comprehensive set of tools that can be used to produce near-optimal DSN designs suitable for a wide variety of applications. This methodology also highlights how our design procedures can be used to both rapidly generate prototype DSN designs to provide insight into the significant aspects of the problem at hand and later refine the design to provide a as near-optimal design as possible.

10. The use of a realistic population size distribution and backbone network design algorithm to design "random" test networks. The random networks produced allow for the assessment of the performance of the design procedures using realistic inputs without the possibility of selecting networks that unduly (dis)advantage one procedure over another. See Section 6.2.2 and first published in [Stac96a] and [Stac96b]. This generation method has allowed us to produce statistically sound measures of the relative performance of the design procedures examined. Most existing network design results in the literature presents results taken from a small (and possibly not truly representative) selection of network examples.

11. A complexity analysis and comparison of the execution time and costs of the heuristic design procedures showing that the clustering procedure provides the best trade off between solution quality and execution time. See Section 5.4 and Sections 6.3 and 6.4.

1.4 Publications Based on this Dissertation

1.4.1 Published

[Stac97c] Chris H. E. Stacey, Tony Eyers, Gary J. Anido, "A Lower Bound for Client-Server Network Design Problems," in *Proceedings of the International Conference on Telecommunications 1997 (ICT97)*, Melbourne, April, 1997.

[Stac97a] Chris H. E. Stacey, Tony Eyers, Gary J. Anido, "A Concave, Link Elimination (CLE) Procedure and Lower Bound for Concave, Topology, Capacity and Flow Assignment (TCFA) Network Design Problems," in *Proceedings of the International Conference on Telecommunication Systems Modelling and Analysis 1997 (ICTS97)*, Nashville, TN 37203, USA, March, 1997.

[Stac96c] Chris H.E. Stacey, Tony Eyers, Gary J. Anido, "A Node Clustering Approach to the Design of Hybrid Access/Backbone Networks," in *Proceedings on the Australian Telecommunication Networks and Applications Conference 1996 (ATNAC'96)*, Melbourne, December, 1996.

[Stac96b] Chris H. E. Stacey, Tony Eyers, Gary J. Anido, "Network Design in a Hybrid Client/Server Peer Network Environment," in *Proceedings of Networks'96*, Sydney, November, 1996.

[Stac96a] Chris H.E. Stacey, Tony Eyers, Gary J. Anido, "Client/Server Network Design." The Institute for Telecommunications Research, University of Wollongong, Technical Report No. TITR-96-1, July, 1996.

[Stac95] Chris H.E. Stacey, Gary J. Anido, H.W. Peter Beadle, Hugh Bradlow, "Multipoint, Multimedia Service Network Dimensioning," in *Proceedings on the Australian Telecommunication Networks and Applications Conference 1995 (ATNAC'95)*, pp. 473-8, Sydney, December, 1995.

[Barn94] Scott A. Barnett, Chris H. E. Stacey, Gary J. Anido, H. W. Beadle, Hugh Bradlow, "A Prototype Information Service Architecture in a Distributed ATM Environment," in *Proceedings of Australian Telecommunication Networks and Applications Conference (ATNAC'94)*, December, 1994.

1.4.2 Accepted for Publication

[Stac97b] Chris H. E. Stacey, Tony Eyers, Gary J. Anido, "A Concave, Link Elimination (CLE) Procedure and Lower Bound for Concave, Topology, Capacity and Flow Assignment (TCFA) Network Design Problems." To appear in *Telecommunication Systems* journal.

2. A Survey of Existing Network Design Techniques

Of the making of books there is no end, and much study wearies the flesh.

- Ecclesiastes 2:12b.

2.1 Introduction

This chapter focuses on classes of network design problem which are related to the design of Distributed Server Networks (DSN's). Good general reviews of network design problems and solution methods can be found in [Boor77], [Magn84], [Mino89], and [Ahuj93] (amongst others).

The Distributed Server Network Design Problem (DSNDP) involves the determination of the number, location and capacity of servers, along with the design of both the access and backbone portions of the network. Determining the number and location of servers along with the allocation of client nodes to them relates to existing *location-allocation* problems. Location-allocation problems take a number of forms, such as the facility location, p-median and hub-location problems, which are discussed in Sections 2.2, 2.3, and 2.4 respectively. The design of the network connecting clients to servers and servers to one another is related to two different types of network topology design problem, centralised and backbone network design, which are discussed in Sections 2.5 and 2.6 respectively. Attempts to solve problems that combine aspects from a number of the sub-problems discussed in this chapter are discussed in the following chapter. Despite the similarity of the DSNDP to some of the problems discussed in this and the follow chapter, to our knowledge the complete design of a DSN has not been addressed in the literature.

For each class of design problems we provide a general description of the type of problem included in the class; discuss how it is similar to the DSNDP; review the solution approaches that have been proposed; and discuss how previous work in the area falls short of providing a complete solution to the DSNDP.

Our conclusions are presented in Section 2.7.

2.2 Facility Location Problems

Facility location problems come in a variety of forms, and are known variously as: *location-allocation*, *warehouse* or *plant location* problems. Given a set of nodes with known demands and locations, the common goal of this class of problems is to determine the number, location and size (capacity) of a number of new facilities to act as supply points for the demand nodes that are assigned to them (the reverse problem, where a set of supply nodes are given and the demand locations are to be determined, is easily formulated). Facilities either have no limit on the amount they can supply, leading to the *Uncapacitated Facility Location Problem* (*UFLP*), or they are constrained, leading to the *Capacitated Facility Location Problem* (*CFLP*). It is often assumed that nodes may have their demand supplied by a number of facilities. Alternately, single source constraints may be introduced to ensure that nodes are supplied by only one facility. If, in addition to single source constraints, facility capacities are also constrained, the problem becomes a *generalised assignment problem* [Klin85].

The cost of solutions to these problems may include both production and distribution costs. Production costs are associated with opening and operating a facility of the desired capacity at each location, while distribution costs are associated with traffic flow between pairs of nodes. If both production and distribution costs are linear functions of capacity, the problem of assigning demand nodes to a known set of facilities can be formulated as a linear *transportation problem* [Sola74] [Boor77]. However, production and distribution costs are more often approximated by concave functions of capacity.

When posed as a warehouse location problem there is often one central supply node which supplies the facilities (i.e. warehouses), which in turn supply the demand

nodes. The introduction of the single central supply sometimes simplifies the inclusion of production costs, and allows the formulation of the problem as a single commodity flow problem. Facility location problems without a single central supply node, can often be modelled as single commodity flow problems with the introduction of a "dummy" node connected to each potential facility location that acts as a global source [Magn84].

Good reviews of facility location and related problems can be found in [Reve70] (note also the comment in [Robe71]), [Srid93] and [Srid95].

An example solution of a facility location problem that employs five facilities, but no central supply node, is shown in Figure 2.1.

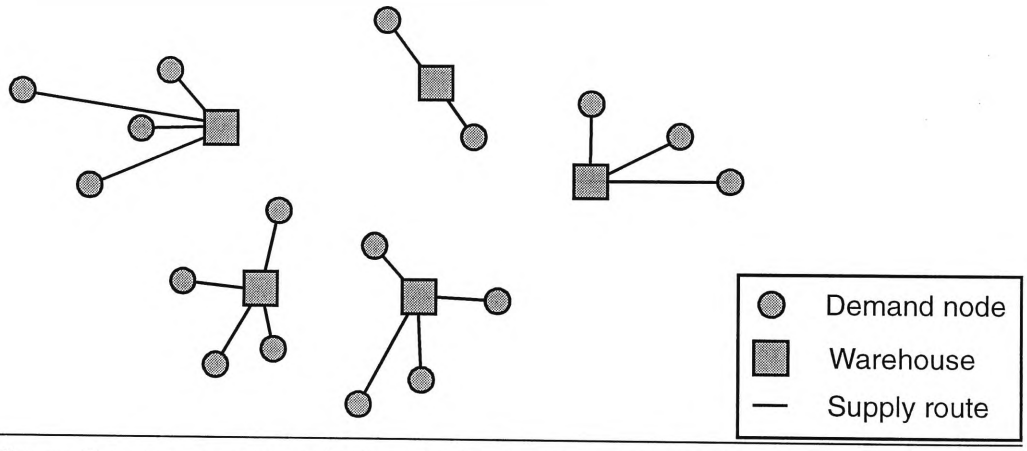


Figure 2.1 An example solution of a facility location problem.

The facility location problem is similar to DSN design in that the objective is to locate servers (i.e. supply nodes, or warehouses) and assign client (i.e. demand) nodes to them. In facility location problems the destination of traffic is known (demand nodes), but its source (the supply node(s)) is not. This means that demand nodes are usually assumed to be directly connected to facilities (which may in turn be directly connected to a central site). The design of the topology of the supporting network is usually not dealt with in these problems. As a result, network performance and reliability constraints are usually not included either. There have been some attempts to include the supporting network topology design in facility location problems. We discuss these types of combined problems in the next chapter.

The remainder of this section describes approaches to solving facility location problems and discusses either how they can be applied to, or how they are not applicable to solving the DSNDP.

2.2.1 Branch and Bound Based Approaches

Branch and Bound based approaches have been employed to solve the UFLP and CFLP. Although none of the algorithms reviewed in this section can be applied directly to the DSNDP they represent a class of algorithms that have long been used in network design. We use a continuous branch-and-bound algorithm (based on [Ryoo95]) in Chapters 4 and 5, a detailed description of which appears in Section 4.4.

In general the branch-and-bound algorithm generates a search tree in which each node represents a partially relaxed version of the original problem. Associated with each node in the tree is the value of the objective function for the solution of the problem represented by the node. At the root of the tree all integer restrictions on the binary variables are relaxed. The solution of the resulting linear program is a lower bound on the cost of the unrelaxed problem. If all of the relaxed binary variables are integer in the solution of the relaxed problem, the problem is solved. Otherwise, two new problems (i.e. nodes in the branch-and-bound search tree) are generated by fixing one of the fractional integer variables first at zero, then at one. The process of creating a number of new problems by fixing a fractional integer variable at each of its possible values is referred to as *branching*. The minimum solution of the two new problems is a new lower bound on the cost of the unrelaxed problem. The process continues by branching on the node which represents the best lower bound. A *terminal* node is a node where the solution to the problem associated with it is infeasible, no branches can emanate from it. The search tree is said to be *bound*, or *fathomed*, at a terminal node. The process terminates when a node is reached where all binary variables are integer and its value is less than or equal to that of any other node. The final node represents the optimal solution to the original, restricted problem. A more complete description of the general integer branch-and-bound algorithm is given in [Nemh88].

Efroymson [Efro66] uses a branch-and bound algorithm to determine the optimum facility locations for a UFLP assuming the demand nodes are supplied by their nearest facility. He does this by associating a fixed charge with opening each facility and formulating a MILP with a binary variable associated with each facility location. The branch-and-bound algorithm from [Efro66] is improved upon in [Khum72] where it is used to solve a standard UFLP. An integer branch-and-bound algorithm is also used in [Crai93] to solve an UFLP with balancing constraints.

In [Sola74], Soland formulates a CFLP in which both production and distribution costs are concave functions of capacity or traffic. Having formulated the problem as a MINLP Soland uses a continuous branch-and-bound algorithm to solve it. The continuous branch-and-bound algorithm is similar to the integer version described above. The difference is that branching is performed by subdividing the domain of the variable of interest rather than fixing it to each possible integer value.

A similar CFLP with concave distribution costs but linear distribution costs is examined in [Khum74]. As in [Sola74] a continuous branch-and-bound algorithm is employed to solve the problem. This algorithm is further developed in [Kell82]. Single source constraints are added in [Klin85] and the continuous branch-and-bound algorithm from [Kell82] employed to solve the problem.

2.2.2 Lagrangian Relaxation Based Approaches

Lagrangian relaxation is a powerful technique that has been used to solve a variety of facility location problems, some of which we review here.

In [Erle78] Erlenkotter examines a UFLP with linear production and distribution costs. Erlenkotter formulates the problem as a MILP, and then relaxes the binary constraints on the binary variable associated with opening a facility at each location and formulates the dual problem using Lagrangian relaxation [Geof74] (see [Fish81], [Fish85] and [Beas93b] for good tutorials on Lagrangian relaxation, and [Shap79] for a survey of its use in solving discrete optimisation problems). The dual problem is solved using a *dual-ascent* procedure which takes advantage of the simple structure of the dual problem. A branch-and-bound algorithm is employed to quantise any linearised binary variables that take up fractional values in the dual solution.

In [Klin86] Klincewicz uses Lagrangian relaxation and Erlenkotter's dual-ascent procedure [Erle78] to solve a CFLP with single source constraints. A similar approach is taken in [Srid93] using sub-gradient optimisation rather than the dual-ascent procedure to solve the dual problem.

In [Louv92] Erlenkotter's dual-ascent procedure [Erle78] is combined with stochastic programming to solve an UFLP when uncertainties are introduced into demands, production and transportation costs.

In [Gao94] both dual-ascent and branch-and-bound procedures are employed to solve a family of UFLPs. Three types of UFLPs are considered: (i) a *single echelon* UFLP, (ii) the *two-echelon* UFLP, and (iii) the *multi-activity* UFLP. The single echelon UFLP is simply a standard UFLP with demand nodes being connected to a central supply node via single layer, or echelon, of facilities. The two-echelon UFLP adds another layer of facilities so that each demand node may be connected to the central site via either one or two facilities. In the multi-activity UFLP there is only a single echelon of facilities but they are able to supply either of two types of commodities demanded from the central site. The problem is then to determine which commodity each facility will supply as well as the number and location of facilities and the assignment of demand nodes to them. This problem is more like the DSNDP than those above, but still does not consider the design of the commodity transportation network.

At first glance it appears the design of the network component of the problem is dealt with in [Chun92] where both dual-ascent and branch-and-bound procedures are employed to solve a two-level hierarchical network design problem. Given a set of user nodes and a set of potential backbone nodes sites, the task is to locate a backbone network (i.e. to determine the number and location of backbone nodes), and assign each user node to a single backbone node. User nodes are connected to backbone nodes in star arrangements, while the backbone network is fully meshed. Although the problem appears in the guise of a network topology design problem, the simple access and backbone link topologies assumed and the absence of any requirements, costs or constraints associated with traffic flows mean that the problem can be formulated as an UFLP. The authors express the problem as a MILP and formulate a linear relaxation of the problem which is solved using Erlenkotter's

dual-ascent procedure [Erle78]. Any factional integer variables are then quantised using a branch-and-bound procedure to arrive at a final solution. The same solution approach is employed in [Kim95], where the problem is extended such that each user node is assigned to two backbone nodes to improve the reliability of the network.

Another formulation of a design problem similar to the DSNDP is provided by Lo and Kershenbaum when they consider the *Star-Star Concentrator Location Problem* (SSCLP) in [Lo89]. In this problem all terminals (demand nodes) are connected to a central site via capacitated concentrators (facilities). In this CFLP, terminals must be assigned to a single concentrator (i.e. node demands must be satisfied by a single source). The costs associated with connecting terminals to concentrators and concentrators to the central site are offset-linear functions of traffic flow. In [Lo89] the authors use Lagrangian relaxation to relax the single source and concentrator capacity constraints. The resulting dual problem is partially solved using sub-gradient optimisation (see [Fish85] or [Beas93b]). After a few hundred iterations little additional improvement is found by the sub-gradient optimisation so it is terminated at that point and an integer branch-and-bound algorithm is applied to the solution produced by the sub-gradient optimisation phase.

Unfortunately, while the SSCLP does examine a two tier structure of facilities, it does not include any significant network design element, as required in the DSNDP.

Lagrangian relaxation is used to solve the SSCLP in [Mirz85]. Having formulated the problem as a MILP, Mirzaian uses Lagrangian relaxation to determine a lower bound on the cost of the optimal solution to the problem. He also develops an approximation algorithm that is able to produce good quality solutions to the original problem, based on the solution of the lower bounding Lagrangian dual problem.

An extension of the capacitated SSCLP is considered in [Pirk88], [Pirk89], [Nara90] and [Nara94]. In these papers, terminals are assigned to multiple facilities to increase the reliability of the network. The constraints associated with assigning terminals to a specific number (or level) of concentrators are relaxed using Lagrangian relaxation and the integer constraints linearised. The resulting problems are solved using specifically developed heuristics that take advantage of the special structure of the problem.

A similar problem to that presented in [Pirk88], [Pirk89], [Nara90] and [Nara94], without the central supply site, is examined in [Ouve94]. As in [Pirk88], [Pirk89], [Nara90] and [Nara94] the multiple assignment constraints are linearised and relaxed using Lagrangian relaxation. However, the resulting dual problem is solved using a general purpose sub-gradient optimisation procedure rather than a special purpose algorithm as in [Pirk88], [Pirk89], [Nara90] and [Nara94].

Barcelo [Barc91] employs Lagrangian relaxation to break the assignment constraints in a CFLP. The resulting relaxed problem is decomposed into two subproblems - an assignment and a knapsack problem. Sub-gradient optimisation is used to solve these subproblems to determine both upper and lower bounds. An integer branch-and-bound algorithm is used to produce feasible solutions from the upper bound.

While it is evident from the literature that Lagrangian relaxation is a powerful technique for solving a variety of facility location problems it has not been applied to a facility location problem of the complexity of the DSNDP. It is also evident from the literature that Lagrangian relaxation based solution procedures rely on some key features of the problem at hand. In Section 2.6.2 we examine whether Lagrangian relaxation is applicable to the DSNDP.

2.2.3 Greedy Search Based Approaches

Greedy search based procedures are another class of solution techniques that have been applied to facility location problems in the past. Greedy search procedures are particularly powerful because they rely on little or no knowledge of the design problem and hence the form of the solution sort. While this "blindness" is probably their greatest strength it is often also their greatest weakness and often limits their usefulness.

In [Spie69] Spielberg examines an UFLP problem and employs both ADD and DROP heuristics for solve it. The ADD heuristic starts with the minimum number of facilities and opens additional facilities one at a time until no further improvement is found. The DROP heuristic is similar but starts with all possible facilities open and closes one at a time. The ADD and DROP heuristics are commonly employed in solving combinatorial optimisation problems as they do not rely on any

special structure of the problem and are simple and robust. ADD, DROP and interchange heuristics are also employed in [Boor77] and [Srid95] to solve a CFLP. In [Tang78] an ADD heuristic is also employed to solve a CFLP in which terminals are connected to multiple facilities. We employ both combined ADD/DROP and ADD-*k* algorithms in Chapter 5 to determine the number and location of servers in the DSNDP (a detailed description of the algorithms is presented in Section 5.3).

2.2.4 Conclusions

The three most significant differences between facility location problems and DSN design are in the areas of network topology design, the interaction between facilities, and the cost functions assumed.

In facility location problems, demand nodes are connected directly to facilities, which may in turn be connected to a central site. In the DSNDP the link topology connecting clients to servers and servers to one another is assumed to have a significant impact on the cost of the final solution. The lack of network topology design in facility location problems means that issues related to network cost, performance and reliability are not addressed.

Secondly, facility location problems assume no interaction between facilities. In contrast, servers do interact in a DSN. The inclusion of inter-facility interaction greatly complicates the problem, so that the solution approaches described in this section which take advantage of special structures in the problem (such as [Erle78], [Lo89], [Nara90], [Nara94] and [Ouve94]) are not applicable.

Finally, to more accurately model the economies of scale that exist in the pricing of network components, the DSNDP assumes that both link and server costs are concave functions of capacity. Most of the solution procedures described here assume linear cost functions, to enable the formulation of linearised relaxations of the original problem. The introduction of concave cost functions removes this option. In [Efro66] and [Spie69], the authors discuss how concave, piecewise-linear facility cost functions may be handled by introducing pseudo-facilities for each capacity segment along with bounds on the minimum and maximum capacities of the pseudo-facilities. However, the complexity of the problem increases rapidly with the number of segments in the cost function, thus making it unsuitable when the cost

function is relatively smooth. Of the solution approaches described, only the continuous branch-and-bound procedure in [Sola74] is suitable for use with continuous concave cost functions.

The differences between traditional facility location problems and the DSNDP outlined above mean that existing techniques used to solve facility location problems cannot be employed to solve the DSNDP. However, we can build on some of the ideas and tools from this area of network design. In particular, as mentioned above, we employ a continuous branch-and-bound procedure in Chapters 4 and 5 to cope with the continuous concave cost functions assumed in the DSNDP. We also employ both combined ADD/DROP and ADD-k algorithms in Chapter 5 to determine the number and location of servers.

2.3 p-Median Problems

The p-median problem determines the optimal location of p facilities on a network which minimises the average distance from the demand points (nodes) to their closest facility. This problem is also known as the *minisum-location problem* since the objective is to minimise the sum of the travel distances [Gavi95]. The p-median problem was shown to be NP-hard in [Gare79].

The p-median problem is clearly similar to the facility location problem. However, unlike the facility location problems discussed above, the p-median problem deals with locations and distances on a network. As in facility location problems the facilities may be capacitated, or uncapacitated. However, unlike most facility location problems the number of medians to be located, p, is specified. If $p = 1$, the problem is referred to as a 1-median problem, if $p = 2$, a 2-median problem, and if $p > 2$, a p-median problem. In addition, the network on which the medians are to be located may be restricted to a tree [Gavi95]. Furthermore, demand points and facilities may be located either at network nodes only, or anywhere on the network [Jook89].

An example solution of a p-median problem, on a tree network, is shown in Figure 2.2.

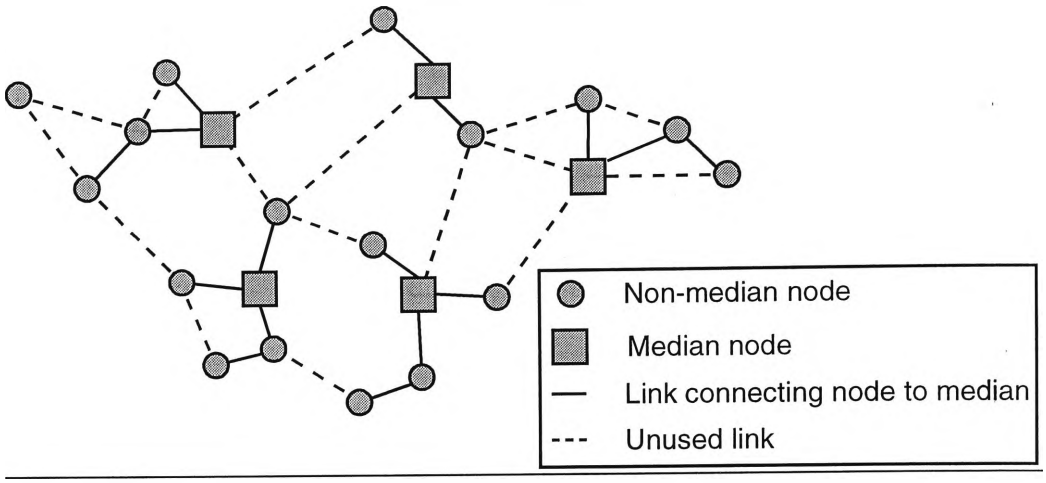


Figure 2.2 An example solution of a *p*-median problem, on a tree network.

Two of the seminal papers on both the *p*-median and *p*-centre problems are [Haki64] and [Haki65]. In [Haki64] Hakimi formulates the 1-median and 1-centre problems. These are generalised to the *p*-centre and *p*-median problems in [Haki65]. In the *p*-centre problem the objective is to locate facilities such that the maximum distance to any single demand node is minimised. In [Haki64] Hakimi proves that a median of a network will always be located at a node of the network. Using that information he formulates the problem as a vertex covering problem and solves it by evaluating all possible combinations of *p* nodes as medians in [Haki65].

In a static network the shortest path distances between pairs of nodes will not change. This means that a static O-D pair cost matrix incorporating link distances rather than geographic distances can be calculated. Once such a matrix is available there is little difference between the *p*-median problem and a facility location problem (see Section 2.2). This means that many of the solutions approaches discussed in Section 2.2 are also applicable to the *p*-median problem [Galv80]. A more comprehensive review of location problems, including the *p*-median problem can be found in [Hand79].

2.3.1 Solution Approaches

An integer branch-and-bound approach is employed to solve a *p*-median problem in [Jaer72]. The problem is formulated as a MILP with binary variables indicating which median each node is assigned to. The authors form a linear relaxation of the problem by allowing the binary variables to take up non-integer values. The integer

branch-and-bound algorithm is used to fix certain variables in each iteration while solving the resulting LP.

Although branch-and-bound approaches to combinatorial optimisation problems are guaranteed to find the optimal solution, they often take an unacceptably long time to do so. In [Galv80], the authors formulate the dual of a linearly relaxed version of the problem. The heuristic developed to solve the dual provides bounds on the optimal solution to the primal problem. The bounds from the dual heuristic are used to reduce the size of the solution space explored by a branch-and-bound algorithm.

In [Naru77] Lagrangian relaxation and sub-gradient optimisation are employed to find a lower bound on the cost of the optimal solution (and hopefully a feasible solution) to the primal p-median problem. The authors mention that if the solution from the Lagrangian procedure is not feasible a branch-and-bound algorithm can be employed to generate one based on the Lagrangian solution. However, this procedure is not guaranteed to find the optimal solution to the problem, and no experience relating to the use of a branch-and-bound in this context is reported in [Naru77].

In [Righ95] a double annealing algorithm is proposed for solving discrete location-allocation problems, such as the p-median problem. The annealing algorithm employed in [Righ95] is called *Mean Field Annealing* (MFA). MFA is an deterministic version of *Simulated Annealing* (SA) [Kirk83] that is particularly suitable for use in solving combinatorial optimisation problems. The decision variables are divided into two groups, with one set determining clusters of nodes assigned to the same median, and the second determining the location of the median within each cluster. A separate annealing procedure is applied to each set of variables with the two procedures being synchronised so that they *saturate* (i.e. settle into states in which all decision variables are *almost* binary and hence define a feasible solution to the problem) at the same time.

Simulated Annealing (SA) is also employed in [Kim96] to solve a series of p-median problems that arise in the context of solving a hierarchical (two-echelon) *Capacitated Facility Location Problem (CFLP)*. The annealing algorithm is started from a number of initial solutions generated by arbitrarily selecting p nodes as medians. The annealing algorithm then generates new solutions by replacing one median node with one non-median node at a time. The authors estimate the size of

the solution space of their p-median problem as $O(N^N)$, where N is the number of nodes in the network. Based on the cooling function used, the execution time complexity of the SA algorithm applied to this p-median problem is estimated as being $O(N^5 \ln N)$. The large number of solutions that must be examined by SA algorithms to arrive at good solutions limits their usefulness in problems where the evaluation of each solution is an expensive process.

In [Jook89] an algorithm is proposed for solving p-median problems where p is small $(p \leq 4)$, and the cost of associating each node with a median is a convex function of distance. The use of a general convex cost function is in contrast with the majority of previous work which assumes only linear, or a very limited range of non-linear cost functions [Jook89]. A particular difficulty introduced through the use of non-linear cost functions is that medians can not be assumed to be located at network nodes (recall that in [Haki64] medians were proven to be optimally located at network nodes under the assumption of a linear cost function). The solution approach presented in [Jook89] involves splitting the network into a number of *tree like segments* (see [Jook89] for details). Each tree like segment is assumed to contain a median and distances between nodes within a segment are linear. The algorithm then enumerates *multisets* of p segments (multisets are sets in which a segment may occur more than once). For each multiset of p segments, the algorithm enumerates the feasible assignments of nodes to medians for the multiset. To determine the optimum location of medians in the multiset of segments, a small convex NLP with linear constraints must be solved for each assignment of nodes. The size of the solution space that must be searched (with a NLP being solved for each solution) restricts the algorithm to cases where p is small $(p \leq 4)$.

In [Gavi95] the authors examine the 2-median problem on tree networks. The special purpose algorithm developed in [Gavi95] is based on first finding the 1-median in the tree and considering the entire tree as being rooted at that node. A *link deletion* algorithm is used to enumerate all possible pairs of sub-trees (*left* and *right*) created by deleting each link in the original tree in turn. 1-medians are determined in each left and right subtree, and the pair of medians that minimise the weighted sum of the distances between nodes and medians is selected as the final solution. A key contribution of the paper is in the way information from previous link deletions

can be employed to make intelligent decisions as to the order in which links are deleted.

2.3.2 Conclusions

There are a number of differences between the p-median problem and the DSNDP. One is that the number of medians to be found is specified for p-median problems, whereas the number of servers is an output of the design process. In addition, medians are assumed not to interact, whereas servers do interact with one another. However, the most significant difference between the two problems is the objective function that determines the location of medians/servers. In the p-median problem the objective is to minimise the weighted sum of the distances between nodes and their medians. Similarly, in the DSNDP the objective includes minimising the cost of links required to connect clients to servers. However, in the DSNDP the cost will also depend on the topology of links employed and the total traffic on (and hence capacity of) each link in the network.

Approaches to the p-median problem are characterised by requiring the enumeration of a large portion of the potential solution space. In [Haki64], [Haki65], and [Jaer72], the entire solution space is searched. [Galv80] attempts to bound the size of the solution space before searching it. The annealing procedures of [Righ95] and [Kim96] are essentially local search techniques that are allowed to climb hills occasionally. Finally, the specialised algorithms developed in [Jook89] and [Gavi95] come down to enumerating sets of possible feasible solutions. The success of enumeration style algorithms relies on their ability to evaluate a large number of potential solutions quickly. This means that while the p-median problem is similar to the DSNDP the solution approaches presented in the literature are not directly applicable to the design of DSN's. This is because (as we discuss in Chapter 3) the evaluation of a solution to the DSNDP is an expensive process, thus ruling out any approach that requires the evaluation of a large number of potential solutions.

2.4 Hub Location Problems

Hub location problems come in a variety of flavours and names, including: *switching centre location*, *hierarchical network design*, and *(interacting) hub location*

problems. The common goal of these types of problems is the design of a system allowing a set of nodes to communicate with one another via a set of *hubs* (sometimes known as *switching centres*).

The hub location problem can be stated as follows: given a set of nodes, their locations, an O-D pair traffic requirements matrix specifying the level of interaction between each pair of nodes and the number of hubs that are to be employed, p, determine the location of hubs and the assignment of non-hub nodes to them to enable the interaction of the non-hub nodes at minimum cost.

Non-hub nodes are connected to directly to hubs, and hubs are directly connected to one another. Links between non-hub nodes are not allowed, so all traffic must be routed via at least one hub. In addition, hubs are often fully meshed ([OKe86], [OKe87], [OKe92], [Klin91], [Ayki92] and others).

The location of hubs may be restricted to a given set of locations, in which case the problem is called a *discrete hub location problem* (see for example [OKe87] and [Klin91]). Alternately, the location of hubs may be unrestricted, allowing them to be located anywhere in the plane on which nodes are located. This is called a *planar hub location problem* (see for example [OKe86], [OKe92], and [Ayki92]).

An example hub based network design is shown in Figure 2.3.

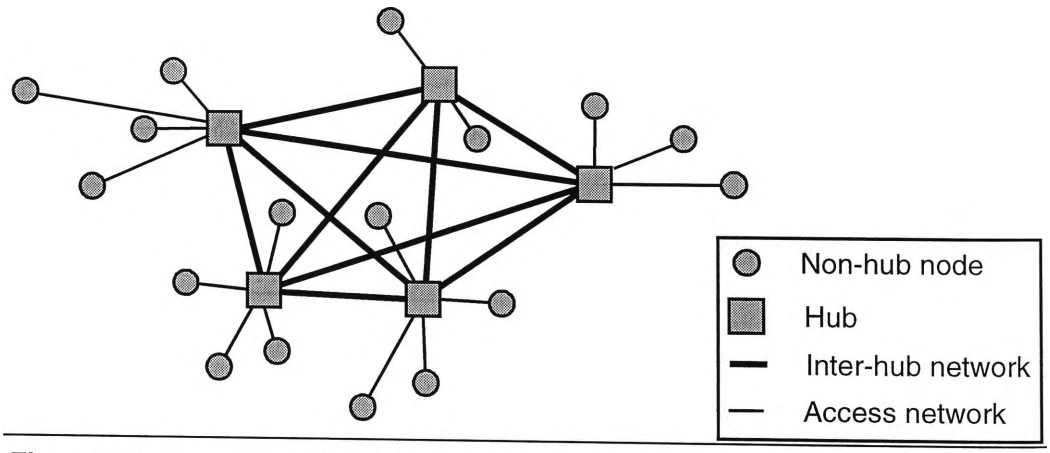


Figure 2.3 An example hub based network design.

The cost of a hub based network is the sum of the costs of the links employed to connect non-hub nodes to hubs and hubs to each other. The cost of a link may be fixed, or be an offset-linear function of their length and/or capacity, where a links capacity is determined by the volume of traffic it carries. In addition, a constant

scaling factor, $0 \leq \alpha \leq 1$, is often used to reduce the cost of inter-hub links to model the economies of scale in the pricing of high capacity links (see for example [OKe86], [OKe87], [OKe92], [Klin91], and [Ayki92]).

The hub location problem is similar to the DSNDP where clients equate to non-hub nodes, while hubs equate to servers which interact with one another. The structure of the networks designed in each case is similar with both consisting of an access portion connecting clients (non-hub nodes) to servers, and a backbone portion inter-connecting servers (hubs).

2.4.1 Exhaustive Search Based Approaches

In [OKe86] O'Kelly examines a simple planar problem where either one or two hubs are introduced into the network (with the designer deciding whether one or two hubs will be used). The solution method proposed in [OKe86] performs an exhaustive search of the possible partitions of the n non-hub nodes into p (where p is the number of hubs to be located) non-overlapping, convex-hull[1] sets. In the case of $p = 2$ there are $(n(n-1))/2$ possible partitions. For each possible partitioning, hubs are located at the centre of mass of each set of nodes and the cost of the solution calculated. The number of possible partitionings of nodes increases rapidly with the number of nodes and hubs to be located. It is interesting to note that the solution method presented in [OKe86] effectively attempts to find a good clustering of nodes around hubs. This is an idea which O'Kelly returns to in [OKe92] and which we apply to the DSNDP in Chapter 5. Node Clustering approaches to the hub location problem are discussed further in the next section.

In [OKe87] O'Kelly examines the problem of locating p hubs (where p is given) at a subset of k given potential hub locations (a discrete location problem). O'Kelly develops two heuristics to solve this problem. Heuristic 1 performs an exhaustive search of all possible combinations of p hubs in k potential locations. For each configuration of hubs, nodes are assigned to their nearest hub. Heuristic 2 also performs an exhaustive search of the hub configuration space. However, for each hub configuration all possible combinations of assigning hubs to their nearest and then second

1. Imagine each node is a nail in a board. The convex hull is the line formed by a rubber band stretched around the outside of all nodes.

nearest hubs are evaluated. Although these heuristics are able to find good solutions, neither is suitable for use on large problems due to the rapid increase in the size of the solution space with the number of nodes and potential hub locations. We examine the complexity of these types of heuristics when applied to the DSNDP in Section 3.4.

2.4.2 Node Clustering and Exchange Based Approaches

Klincewicz also examines the discrete hub location problem in [Klin91]. Klincewicz compares O'Kelly's first heuristic from [OKe87] (described above), with two new heuristics, one based on a node clustering heuristic developed in [Monm86], and the other on a node exchange heuristic.

The node clustering heuristic selects the p nodes that are the greatest concentrations of source and destination traffic as *seed* nodes. Clusters are built around the seed nodes to maximise the weighted sum of a common traffic measure with the cluster (measured in units of traffic interchanged with nodes assigned to the same cluster), and a distances measure to the centre of mass of the cluster (measured as the inverse distance from the node to the centre of mass, so that the closest cluster centre has the largest measure). Once all assignments have been made they are re-evaluated one at a time to determine if any further improvement can be made. Finally the node closest to the centre of mass of each cluster is designated as the hub to which all nodes in the cluster are assigned.

The exchange heuristic starts with a similar set of seed hubs and assigns non-hub nodes to the seed hubs using a procedure similar to that employed by the clustering heuristic. The exchange heuristic attempts to improve on the current best set of hub locations by evaluating all sets of hub locations that differ in one or two hub locations. Locations not in the current set are *exchanged* with those that are, based on improvement in the objective function due the hub exchange without any reassignment of non-hub nodes. Any improvement would be increased by the subsequent reassignment of non-hub nodes.

In effect the exchange heuristic starts by forming clusters of nodes using the seed nodes as the basis for the distance measure used to form the clusters (rather than the centre of mass, as is done in the clustering heuristic). Then rather than perturbing

the clusters formed by reassigning non-hub nodes, it perturbs them by moving the hubs around which each cluster was formed. Moving the hub may then motivate a reassignment of nodes amongst clusters (hubs).

Both the heuristics in [Klin91] assume that all nodes are potential hub locations. However, they could both easily be modified to handle the case where not all nodes are potential hub locations.

Klincewicz found that in general the exchange heuristic is more robust than the clustering heuristic and performed almost as well as the enumeration based heuristic from [OKe87]. Unfortunately, the exchange heuristic relies on rapidly evaluating the local improvement measure which is based on the interaction between non-hub nodes. In a DSNDP there is no interaction between client (non-hub) nodes[2] and the evaluation of each exchange of hubs is a complex task involving the solution of a Concave TCFA problem.

The planar hub location problem is addressed by Aykin in [Ayki92]. He develops a heuristic similar to the exchange heuristic from [Klin91]. Akyin also alters the distance metric to examine the problem of locating hubs on a sphere rather than a plane. While this alteration more accurately models the location of hubs on the surface of the Earth, it complicates the problem making the assignment of nodes to hubs a non-convex problem.

In [SK94] a hub exchange heuristic is augmented by the use of a *tabu list* creating a *tabu search* heuristic for an uncapacitated, discrete, hub location problem. Tabu search ([Glov89], [Glov90]) is an approach to overcome the problem of a search becoming trapped in local minima in problems with non-convex objective functions. The algorithm maintains a tabu list of recently visited solutions. In each iteration of the search, the algorithm is prohibited from revisiting any solution contained in the tabu list. Rather than stopping the search when a minima is reached, the search continues moving to the least cost solution not in the tabu list (the search is terminated after a specified number of iterations, or when no improvement has been

2. Traffic from a given client node which transits through another to its destination does not count as non-hub (i.e. client-client) node interaction as it is a by-product of client-server interaction. In a real network there may be non-hub node interaction due to other services. For simplicity we assume traffic from different services is segregated through the use of Virtual Service Networks (VSN). The inclusion of client-client traffic in the DSNDP is discussed in Section 6.2.4.

found for a specified number of iterations). In this way the search is encouraged to "climb out of" local minima, thereby having a better chance of finding the global minima. The heuristic employed in [SK94] starts by locating the specified number of hubs at nodes with the highest levels of interaction. The remaining non-hub nodes are assigned to their nearest hub. An exchange heuristic is then used to swap hub and non-hub nodes in an attempt to find better solutions. The tabu list is used to ensure that recently visited configurations are avoided.

O'Kelly returns to the planar hub location problem in [OKe92] where he employs a node clustering approach to the problem. The objective is to cluster n nodes into p groups, so that the sum of the squared distances between the nodes in each cluster and the centre of mass of each cluster is minimised. The mass of each node, used to calculate the centre of mass of a cluster, is determined by the intra and inter-cluster interactions *of* each node.

A node clustering approach is also employed in [Diri77] to solve a hierarchical hub location problem. As in standard hub location problems client nodes are connected directly to hubs. However, rather than hubs being connected to one another they are connected to another, higher, layer of hubs, and so on until there is only one hub, thus creating a hierarchical tree of star designs at each level. Another significant difference between the problem dealt with in [Diri77] and the standard hub location problem is that the number of hubs to be employed is not specified in [Diri77]. Rather the clustering algorithm determines the best number of hubs to employ at each level of the network. This problem is similar to the DSNDP in that it allows the algorithm to determine the number of hubs employed. However, it still assumes a very simple tree of stars network topology, rather than considering how traffic could best be combined routed between nodes as in the DSNDP.

In [Ayki94] Aykin employs branch-and-bound, an exchange heuristic and Lagrangian relaxation, to design a capacitated hub-and-spoke network. This problem extends the standard hub location problem by allowing non-hub nodes to be both connected to multiple hubs, and directly connected to one another. Thus traffic may flow directly between highly interactive nodes, or may travel via the hub network. If traffic does travel via the hub network it is assumed to be routed via either one or two hubs to its destination. The design problem is considered in two parts, the loca-

tion of hubs, and the routing of traffic via direct, one-hub stop and two-hub stop routes. Both a discrete branch-and-bound and a greedy interchange (exchange) heuristic are considered to solve the hub location problem. The MILP routing problem is solved using Lagrangian relaxation and subgradient optimisation. The planar version of this problem is examined in [Ayki95]. This problem is similar to the DSNDP in that it examines the design of a two tier network topology with traffic flowing between both clients and servers (hubs) and from one hub to another. However, the major differences are in the simple network topology and link cost functions considered. Client nodes are connected directly to hubs, rather than via one another as they may be in, for example, a tree shaped access network. Secondly, the linear link cost functions used allow the routing problem to be solved relatively easily. The inclusion of concave non-linear link (and hub) cost functions, as in the DSNP, would render the solution procedure ineffective.

2.4.3 Conclusions

There are a number of differences between the hub location problem and the DSNDP.

The first difference is the type of interaction between nodes in each problem. A hub network design is driven by interactions between non-hub nodes. In contrast, in a client-server environment, the design is driven by interactions between clients and servers, and between servers. This means that the improvement measures employed in the hub location algorithms to determine the locations of hubs and assignment of non-hub nodes to them are not applicable for use in the DSNDP.

The second area of difference is in the network link topology design. The hub location problem does not include any link topology design. Non-hubs nodes are typically connected to hubs in a star topology and hubs are inter-connected by a full mesh of links. Related to this is the fact that the (offset-)linear link cost functions used in hub location problems cannot capture the economies of scale in link capacity pricing. The economies of scale in link pricing and the topology of links will have a significant impact on the cost of a network design. In addition, hub location problems do not consider any performance or reliability constraints associated with the design of the network.

Finally, the number of hubs to be located is an input to most existing hub location algorithms. One of the tasks posed in the DSNDP is the determination of the optimal number of servers. To our knowledge, the only approach that addressed this issue in the context of hub location problems is the clustering based procedure in [Diri77].

Most of the existing solution procedures for hub location problems are not appropriate for determining the number of servers and assignment of clients to them in a DSNDP because they rely on either (i) the linear cost functions to formulate MILP's that can be solved as part of the solution procedure, or (ii) on evaluating a large number of potential solutions to the problem. However, the clustering based procedures of [OKe92] and [Diri77] show potential as they cluster nodes together such that intra-cluster interaction is maximised while minimising inter-cluster interaction, without relying on either the form of the cost functions, or evaluating a large number of potential solutions. Node clustering based procedures are considered further in the next chapter and we develop our own node clustering based procedure for the DSNDP in Chapter 5.

2.5 Centralised Network Design

Due to the complexity of design problems involving both access and backbone portions of a network, previous work has focused on splitting the problem in two, and concentrated on the design of access and backbone portions separately [Boor77], [Gavi91], [Pirk92]. In this section we review previous work on *centralised network design*, that is, the problem of designing the access portion of the network that connects a collection of end user nodes with a single central location.

Centralised network design is also known as *access network design*, or the *Capacitated Minimum Spanning Tree (CMST)* problem. In CMST problems the network is restricted to be a tree with links constrained to a given capacity. The objective of the CMST problem is to determine a link topology that defines a tree of minimum total length while satisfying constraints that limit the total traffic and/or the number of nodes in any subtree rooted at the central site [Kers80]. Such subtrees may be considered as multipoint lines in a centralised network.

In the classic CMST problem a "distance" matrix specifying the cost of connecting each pair of nodes is given. A key feature of this cost matrix is that the cost of each link is independent of both the other links included in the solution and the volume of traffic that it carries. This contrasts significantly with the link cost function employed in both backbone network design problems and DSNDP's.

In its purest form, the design of a centralised network involves determining the topology, and capacity of links connecting client nodes to central sites given the location of the central sites and the assignment of client nodes to them (see [Mino89] [Gavi91] for a good review of these types of problems).

Some centralised network design research combines the location of concentrators with the topological design of the network. This area is examined in Section 3.3.1.

Centralised network design forms a part of the DSNDP, as client nodes must be connected to their associated servers. In previous work on combined network design problems (see Section 3.3) that involve both access and backbone networks, user nodes are often assumed to be directly connected to backbone nodes, creating a star topology in the access network. An example centralised network design is shown in Figure 2.4. It is clear that a centralised network design, such as the example in Figure 2.4, is significantly different from that which would be produced by connecting all user nodes directly to the central site (thus creating a star topology). More importantly, the total cost of the designs would also be significantly different. This is particularly so if the link cost function is a concave function of link capacity as a centralised design will attempt to employ fewer, higher capacity links than a star network design. The fact that the design of the access portion of DSN's cannot be ignored, or assumed to be based on star topologies, leads us to examine previous approaches to solving centralised network design problems.

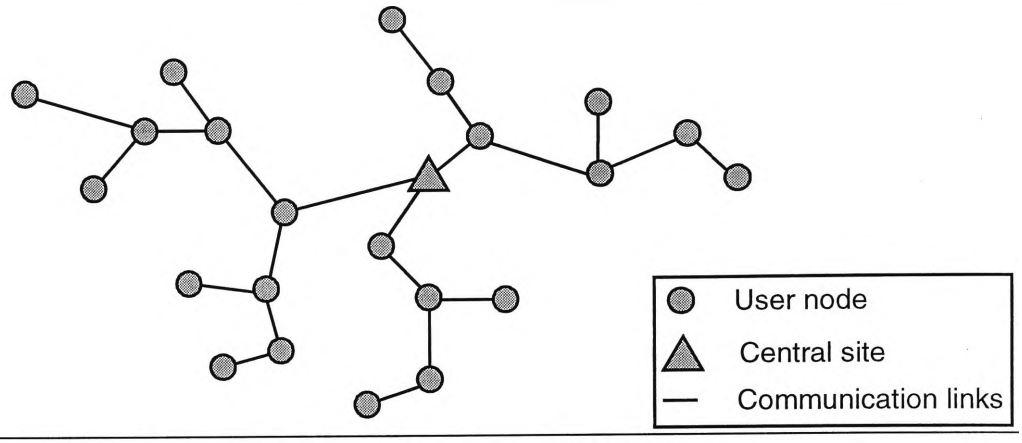


Figure 2.4 An example centralised network design.

2.5.1 Greedy Search Based Approaches

Kershenbaum provides a good review of the early work on the CMST problem in [Kers80]. He observes that optimal solutions can be generated by branch-and-bound procedures. Unfortunately, the execution times of the branch-and-bound algorithms are generally exponential in the number of nodes. This means they are not practical as general solution procedures so he turns instead to greedy search based approaches.

Kershenbaum also claims (in [Kers80]) that most existing (at that time) heuristic design procedures (see references in [Kers80]) for the CMST problem follow a similar, greedy, pattern of operation. They start with each node connected directly to the central site. Links between pairs of nodes are then considered. If the introduction of a link reduces the cost of the solution without violating constraints it is included and one of the links to the central node removed. This process continues until no further improvement can be made. A variety of heuristics are available to guide the consideration of links and determine the benefit of introducing a new link. The solutions generated by these greedy heuristics, which Kershenbaum refers to as *First Order Greedy Algorithms (FOGA's)*, are claimed to be within 50% of optimum [Kers80].

In [Kers80] a *Second Order Greedy Algorithm (SOGA)* employs a FOGA iteratively to produce solutions approximately 5% better than those produced by the FOGA's alone. A key observation behind Kershenbaum's SOGA procedure is that one of the common differences between optimal and heuristic solutions is the inclusion of

links from the Minimum Spanning Tree (MST) in the optimal solution, and their exclusion from the heuristic solutions. The SOGA therefore starts by determining both the optimal MST and a heuristic solution to the CMST problem (generated by a FOGA). Subsets of links in the MST but not the heuristic solution are added to the heuristic solution and the FOGA reapplied in an attempt to improve the solution.

In [Gavi85a] Gavish formulates the CMST problem as a zero-one Integer Programming (IP) problem. The formulation is based on constraints derived through the solution of bin packing problems used to determine the number of nodes on a multidrop line (i.e. branch). The LP relaxation of the integer program can be used as a lower bound on the optimal solution to the CMST problem. The large number of bin packing constraints in the problem formulation leads Gavish to implement an *augmented Lagrangian* procedure to solve the relaxed problem. The augmented Lagrangian procedure is based on solving a *Degree Constrained MST (DCMST) problem.* The cost matrix of the DCMST is determined by adding a subset of the bin packing constraints, multiplied by appropriate Lagrangian multipliers, to the objective function. The multiplier values for the Lagrangian problem are first computed using a dual ascent procedure, and then updated by a subgradient optimisation procedure. Although the algorithm presented in [Gavi85a] is specific to the CMST problem it provides an example of how a variety of optimisation procedures can be combined to solve a complex problem which is cannot be handled by a single approach alone. In Section 3.4 we examine solution approaches to the DSNDP and determine that it too benefits from the application of a variety of complimentary procedures.

In [Gavi91] Gavish revisits the CSMT problem and formulates it as both an IP problem with bin packing based constraints (as in [Gavi85a]) and as an IP, multicommodity flow problem. The linear relaxation of the bin packing based formulation is shown to be stronger than the linear relaxation of the multicommodity flow formulation.

Gavish [Gavi91] also describes a *Q Iterated Tour Partitioning (QITP)* algorithm for the CMST problem. The QITP algorithm has the advantage that it is able to provide both upper and lower bounds on the optimal solution of the CMST problem. In

addition, the algorithm has a constant worst case bound on the error between the lower and upper bounds on the cost of the optimal CMST.

Unfortunately, the bounds produced by the QITP algorithm are too large to be of practical use in large problems. In practice faster heuristics, without guaranteed worst case performance, are employed. However, the bounds produced by the QITP algorithm can be used to assess the quality of the solutions produced by the heuristics. This is an idea which we return to in Chapter 4.

In Section 4 of [Gavi91], Gavish provides a concise review of the existing heuristics for solving the CMST problem, and describes a *Parallel Savings Algorithm (PSA)* for the CMST problem. As described above (see discussion of [Kers80] in this section), many existing heuristics operate by greedily merging a single pair of branches in each iteration until no further improvement is possible. The PSA algorithm also operates by starting with a star topology and merging branches. However, the PSA algorithm performs multiple branch merges in each iteration. The cost benefit of each potential merging of pairs of branches is evaluated, and a maximum cost merging problem is solved to determine the subset of merges to be performed. While the PSA algorithm takes 2 to 3 times longer than existing algorithms, it produces solutions whose cost is in the order of 2 to 4 percent cheaper (a significant improvement given the large cost of modern communications networks).

The PSA algorithm is modified in Section 5 of [Gavi91] to handle multicentre tree networks. The *Multicentre Capacitated Tree Problem (MCT)* problem involves designing a set of CMST's that connect nodes to a set of "central" locations. This problem is similar to the DSNDP. The central locations can be considered as backbone nodes, or servers, with client nodes connected to them by CMST's. The MCT algorithm presented in [Gavi91] assumes the number and location of the backbone nodes are known. If they are not, an uncapacitated facility location problem (UFLP) is solved to determine the number and location of backbone nodes. The MCT algorithm starts with all nodes in separate branches, unconnected to any backbone node. In each iteration, all pairs of branches are considered for merging. If a potential merging of branches does not violate the capacity constraint on a single branch, the assignment of the merged branches to each backbone node is evaluated. A matching problem is solved to determine the subset of branch mergers, and the assignment of

branches to backbone nodes, to be implemented in each iteration. Gavish reports that the MCT PSA algorithm gave improvements in cost of up to 3.43% over an Esau-Williams algorithm modified to solve the MCT problem.

In Section 6 of [Gavi91], solution procedures for *Minimal Cost Loop (MCL)* problems are summarised. MCL problems are similar to MST problems in that a number of terminals are to be connected to one, or more, central sites. The difference is that the link topology employed is a loop rather than a tree. This type of access network link topology would be applicable in an environment where a networking technology such as IBM's Token Ring, or FDDI, was to be used. The reader is referred to [Gavi91] for more details.

Section 7 of [Gavi91] deals with the *Telepak* problem. The problem is described, and a few of the early papers on the problem are referenced, but no solution procedures are described. The Telepak problem extends the CMST problem by including the selection of a capacity for each link. In addition, the cost of each link is dependent on the traffic load it carries (recall that in the CMST problem link costs are independent of traffic load). The inclusion of capacity based link costs and link capacity selection make this problem interesting as these aspects are also included in the DSNDP.

The Telepak problem is also addressed in [SG96]. The authors modify the formulation of the problem from [Gavi91] to allow inactive nodes to be included in the network. The formulation in [Gavi91] assumes that all nodes generate traffic to be delivered to the central site. A linear relaxation of the problem is formulated, and solved, along with the primal problem, using the commercial mathematical programming package LINDO. The results show that the linear relaxation produces solutions whose costs are within 9% of the optimal solution to the primal problem.

A multiple centre Telepak problem is considered in [Mate96]. The number and location of the central nodes is given, as is the assignment of terminals (referred to as remote nodes) to central nodes. This means that the problem is in effect simply a number of separate single centre Telepak problems. The problem is formulated as a MILP and solved using a Lagrangian relaxation based heuristic. Sub-gradient optimisation is used to update the Lagrangian multipliers. The duality gap between the heuristic solutions and lower bound is shown to be between 20% and 30%.

The single centre Telepak problem is also considered in [Lee96] where it is again formulated as a MILP. A LP relaxation of the problem is solved using a cutting-plane algorithm from which feasible solutions are generated using a branch-and-bound algorithm.

2.5.2 Conclusions

Of the centralised network design problems reviewed in this section the Telepak [SG96], [Mate96] and MCT [Gavi91] problems are most closely related to the access network design portion of the DSNDP.

The MCT problem addresses the issue of assigning user nodes to backbone nodes, but relies on solving a *Facility Location Problem (FLP)* to determine the number and location of backbone nodes. Thus it ignores network topology design and any interaction between nodes, both of which are included in the DSNDP. In addition, the assumption that link costs are independent of traffic load simplifies the problem. The use of a traffic load based link cost function, as in the DSNDP, would greatly complicate CMST style problems. The inclusion of this feature is mentioned as an area for future work in [Gavi91]. To our knowledge no further work in this area has appeared in the literature.

While the Telepak problem includes traffic load based link cost functions, existing algorithms rely on the use of offset-linear functions. This allows the problem to be formulated as single commodity flow MILP's which are easily relaxed and solved. However, link cost functions are assumed to be non-linear in the DSNDP thus ruling out the use of MILP based solutions. Unlike the CMST and MCT problems the Telepak problem also allows for the selection of link capacities. However, the number of possible link capacities allowed is kept small in the current studies. This is because the problem size increases rapidly with the number of possible link capacities.

While assuming a small set of possible link capacities is reasonable for most current network technologies the advent of ATM invalidates this assumption. In an ATM environment virtual channels (and hence links) can be assigned almost any capacity, up to the limit imposed by the underlying transmission system. This means that the number of possible link types (capacities) may be very large. In this environment

existing Telepak network design procedures would be limited to designing relatively small networks.

Neither the CMST nor Telepak problems include performance or reliability constraints on the network design. The inclusion of these constraints, as in the DSNDP, complicates the problems and requires the development of new solution procedures. Again, the inclusion of performance and/or reliability constraints is discussed as an area for future work in [Gavi91]. To our knowledge no further work in this area has appeared in the literature.

2.6 Backbone Network Design or the TCFA Problem

The *Topology, Capacity and Flow Assignment (TCFA)* problem, as presented by Kleinrock and Gerla in [Klei76] and [Gerl77], often appears as a subproblem in larger network design problems, such as the DSNDP. A communications network is assumed to be modelled by a graph in which the nodes represent sources or sinks of traffic and edges represent links of the network. Given an origin-destination (O-D) node pair traffic requirements matrix specifying the traffic requirements between all pairs of nodes, the TCFA problem determines which links to include (the topology), their capacity and the routing of traffic in the network. The objective is to find a minimum cost solution which supports the required O-D traffic requirements while satisfying all constraints. Following [Klei76], [Gerl77], [Mino89], [Gers90] and others, we assume that link cost functions are concave with respect to link capacity. This is to capture the effect of economies of scale in link capacity pricing. We refer to a TCFA problem that involves concave link cost functions as a *Concave TCFA* problem.

In the design of DSN's the Concave TCFA problem arises once the location of servers and assignment of clients to them has been determined. Using this information an O-D traffic requirements matrix can be generated and a TCFA algorithm employed to design the network topology connecting clients to servers and servers to one another.

For completeness, an example backbone network design is shown in Figure 2.5.

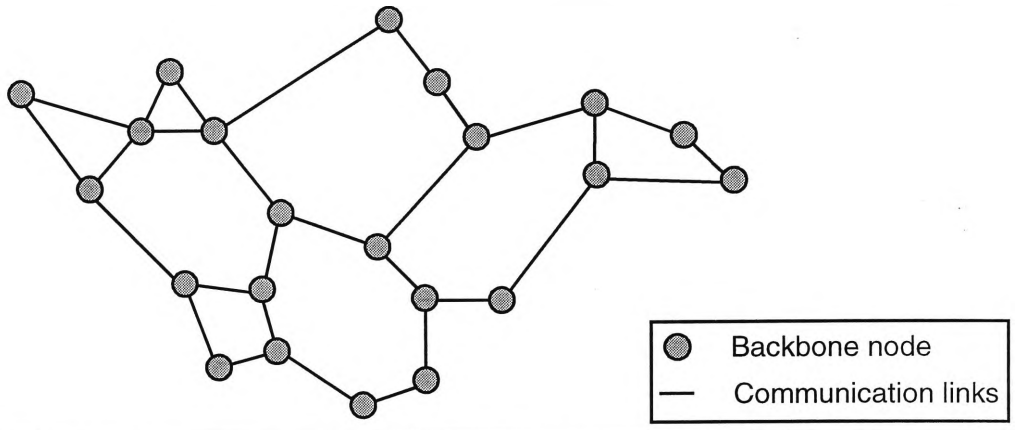


Figure 2.5 An example backbone network design.

Many variants of the TCFA problem, and associated solution procedures, have appeared in the literature. The key approaches are reviewed here, and their applicability to the DSNDP discussed.

2.6.1 Decomposition Based Approaches

Due to the complexity of the TCFA problem early work in the area [Klei76] [Gerl77] [Boor77] focused on decomposing it into a number of related subproblems: optimal capacity assignment, optimal routing and optimal topological design. In [Klei76] and [Gerl77] Kleinrock and Gerla assumed link cost functions to be linear or concave to reflect the effects of economies of scale in the pricing of link capacity. By using a flow deviation technique (from [Frat72]), linearisation of concave link cost functions, and continuous approximations to discrete capacity variables Kleinrock and Gerla develop a *Concave Branch Elimination (CBE)* procedure to solve the TCFA problem. A similar, design procedure called *Cut-Saturation (CS)* is also presented in [Klei76] and [Boor77]. Like the CBE procedure it employs the flow deviation technique from [Frat72] to route traffic in the network. The CBE procedure is shown to be superior to the CS procedure in [Klei76].

Ng and Hoang [Ng87] extend Kleinrock and Gerla's work by modelling the network as a collection of *m*-M/M/1 (rather than M/M/1) queues to make the flow deviation search space strictly convex. This reduces the computational complexity of the design procedure but relies on the use of a stepwise fixed-charge link cost function (i.e. link cost is stepwise-linear with capacity with an additional fixed charge associated with employing a link). The approaches of Kleinrock and Gerla [Klei76]

[Gerl77] have also been employed to examine networks that include widely varying link delay characteristics [Huyn77], and dynamic capacity allocation through the inclusion of temporary leased lines [LeBl90].

2.6.2 Mathematical Programming and Lagrangian Relaxation Based Approaches

Gavish *et al.* [Gavi86b] [Gavi89] [Gavi90] [Gavi92a] treat the TCFA problem as a whole and develop a design procedure using a Lagrangian relaxation of a Mixed Integer Linear Programming (MILP) formulation of the problem. Gavish *et al.* assume that links are available in a small number of capacities, or types. The cost of each link type is modelled by a fixed-charge cost function. The authors assume that the price of link types is concave with respect to capacity. This produces a stepwise linear concave link cost function. The Lagrangian relaxation based solution procedure employed by the authors relies on the number of possible link capacities being small ([Gavi92a] assumes only one link type, while [Gavi90] assumes at most 20). In [Gavi90] an exhaustive search of the possible link capacities is used to determine the capacity of each link.

Assuming a small set of possible link capacities is reasonable for most current network technologies, but the advent of ATM invalidates this assumption due to the flexibility with which capacity may be assigned to virtual links in an ATM network. Thus the number of possible link types (capacities) may be very large, which means that the execution time of an exhaustive search of possible link capacities limits the usefulness of Gavish *et al.*'s TCFA algorithm to designing relatively small networks.

In addition, rather than placing a constraint on the maximum average queuing delay in the network, Gavish *et al.* incorporate the cost of delay into the objective function of their formulation of the TCFA problem. This simplifies the problem by effectively relaxing a non-linear constraint into the objective function. However, it complicates the task of the designer who has to determine the relative cost of delay in "dollar" terms.

Magnanti and Wong [Magn84] also formulate a fixed-charge (i.e. offset linear link cost function) network design problem as a MILP. However, their formulation inte-

grates only routing and topology design and does not consider link capacity allocation. In [Bala89], Balakrishnan, Magnanti and Wong examine the fixed charge uncapacitated network design problem, which is similar to the TCFA problem, without link capacity constraints. Again, they assume fixed charge link cost functions and formulate the design problem as a MILP. Although the dual-ascent solution procedure employed is able to find near optimal solutions it relies on the linear link cost function. It is not applicable to the DSNDP because the DSNDP uses concave link cost functions.

2.6.3 Greedy Link Elimination Based Approaches

Gersht and Weihmayer [Gers90] formulate the TCFA problem following Kleinrock and Gerla but develop a design procedure based on the greedy elimination of links. In addition, Gersht and Weihmayer treat the problem as a whole (rather than decomposing it as Kleinrock and Gerla do). Gersht and Weihmayer's algorithm assumes only that link cost functions are concave and continuous, without relying on the exact form of the function. Gersht and Weihmayer claim that their procedure produces network designs whose cost is within a few percent of those produced by much more sophisticated algorithms in use within IBM. The most significant problem with Gersht and Weihmayer's greedy link elimination procedure is its execution time. In each iteration of Gersht and Weihmayer's procedure, the effect of every possible single link elimination on the cost of the network is evaluated. The link whose removal maximises the improvement in network cost is eliminated, the affected traffic is rerouted and new link capacities are assigned. The process is repeated until no more links can be eliminated without increasing the cost of the network. The process of evaluating each possible single link elimination in each iteration makes the procedure computationally very expensive. Gersht and Weihmayer acknowledge the need to reduce the execution time of their procedure and discuss the idea of eliminating multiple links in each iteration without developing any specific method to do so.

A simple greedy link elimination procedure is presented by Grout in [Grou87]. Like Gersht and Weihmayer, Grout's procedure eliminates only one link per iteration. Unlike Gersht and Weihmayer, Grout does not include any performance or reliability constraints in his procedure. Moreover, Grout's evaluation of the best link to

eliminate in each iteration is more complex than Gersht and Weihmayer's. Grout suggests determining the best sequence of S links to eliminate (where the elimination of the Sth link is conditional on the elimination of the S-1th link and so on). However, rather than eliminating S links in each iteration, Grout eliminates only the first link in the best sequence found. Grout's procedure operates in much the same way as a computer chess program may search a tree of possible sequences of moves to determine its next move. The time required to determine the best sequence of S links to eliminate grows rapidly with the depth of the link elimination search performed. This means that the execution time of Grout's procedure will be considerably greater than Gersht and Weihmayer's procedure, without including performance or reliability constraints. Thus Grout's procedure does not appear to provide any benefits over the procedure proposed by Gersht and Weihmayer.

An *accelerated greedy link elimination algorithm* is presented in [Mino89]. In [Mino89] Minoux starts with an algorithm much like Gersht and Weihmayer's with one significant difference. When a link is eliminated in Gersht and Weihmayer's algorithm the traffic flowing on it is rerouted from source to destination. In Minoux's algorithm all the traffic that the eliminated link was carrying is diverted onto the single, cheapest path between the end points of the link. This may result in what is sometimes known as *trombone routing*, an example of which is illustrated in Figure 2.6.

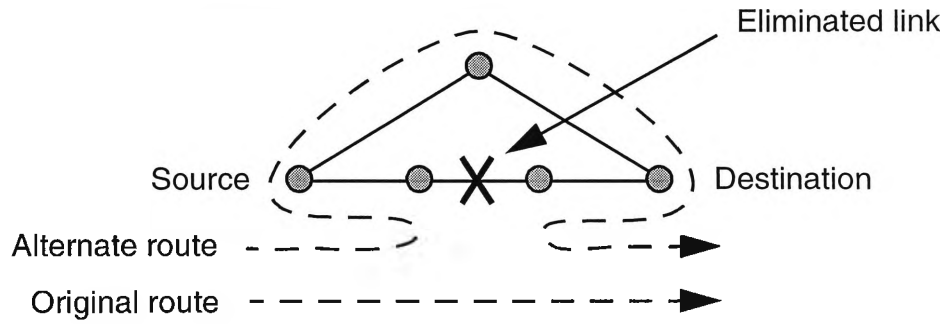


Figure 2.6 An illustration of trombone routing in operation.

The possibility of trombone routing in [Mino89] means that the final routing of traffic will most likely not follow the shortest path between pairs of nodes. This means that the capacities assigned to links will not be suited for use in an operational envi-

ronment in which a shortest path routing strategy (or one that approximates shortest-path routing in the long term) is employed.

The use of this type of rerouting also means that the traffic on a link can be guaranteed not to decrease (note that if the traffic is rerouted from source to destination in Figure 2.6, the traffic on the links either side of the link eliminated would decrease). This leads Minoux to postulate that the incremental increase in network cost due to the removal of a link will never decrease from one iteration to the next.

This can be seen to be true in the context of linear, or offset-linear, link cost functions (a proof is provided in [Mino89]). With the linear link cost functions the incremental cost associated with deviating traffic onto an alternate path is independent of the volume of traffic on the links in the alternate path. This, combined with the fact that the volume of traffic to be deviated cannot decrease, means that the incremental increase in network cost associated with removing a given link cannot decrease. Minoux claims that while it is possible for the incremental cost to decrease in the context of concave link cost functions it rarely does so.

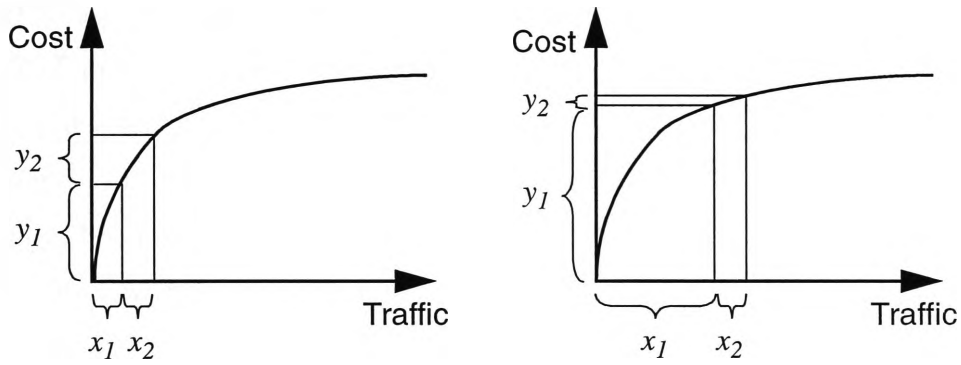
Minoux proposes two acceleration techniques that rely on the incremental increase in network cost associated with the elimination of a link being a non-decreasing function of the algorithm iteration.

The first acceleration technique is based on the observation that once the incremental increase in network cost associated with the removal of a link is positive, it will never (or very rarely) become negative again. This means that the algorithm can be accelerated by marking those links with positive incremental costs so that they are not considered for removal in future iterations of the algorithm.

The second acceleration technique is based on the observation that while the incremental cost associated with the removal of each link must be calculated in the first iteration of the algorithm, only a few incremental costs need to be calculated in subsequent iterations. Firstly, consider that in each iteration of the algorithm the link with the minimum incremental cost will be removed (assuming it is negative). Secondly, remembering that the incremental cost associated with all other links must either remain the same, or increase, the algorithm needs to recalculate new incremental costs associated with links only until it has a new current minimum cost. In

the best case the incremental cost for the second best link would be recalculated and found to still be the minimum of all remaining links. If another link became the minimum then the incremental cost of its removal would be recalculated, and so on until the minimum does not change upon recalculation. Employing this approach greatly reduces the number of link removals that have to be considered in each iteration of the algorithm.

A key claim made by Minoux is that in the context of concave link cost functions, the incremental cost associated with the removal of a link rarely decreases. Our experiments have shown that it is in fact quite common for the incremental cost, due to the removal of a given link, to decrease when concave link cost functions are employed. A simple illustration of how this may happen is provided in Figure 2.7.



(a) Iteration i (b) Iteration $i + n$

x_1 - traffic already on path.
x_2 - traffic deviated onto path due to link removal.
y_1 - cost of traffic already on path.
y_2 - cost of traffic deviated onto path due to link removal.

Figure 2.7 An illustration of how the incremental increase in network cost due to the elimination of a link can increase when concave link cost functions are employed.

Consider the curve in Figure 2.7 (a) which represents the cost of the alternate path employed when a given link is removed from the network. Assume that in iteration i of the algorithm there is already traffic, x_1, with an associated cost, y_1, on the path before the removal of the link. The removal of the link causes additional traffic, x_2, with an associated cost, y_2, to be routed onto the path. At some later time, in iteration $i+n$, where $(n \geq 1)$ (Figure 2.7 (b)), the same link is being considered for

removal. This time there is more traffic already on the alternate path, x_l, however, there has been no increase in the traffic on the link being considered for removal. As can be seen in Figure 2.7 (b) the volume of traffic has not decreased anywhere (as required by Minoux), but the incremental cost associated with removing the link has decreased. Our experiments have shown that this occurrence is relatively common. This means that, contrary to what Minoux claims, the two acceleration techniques proposed in [Mino89] cannot be employed when link cost functions are concave. The DSNDP assumes concave link cost functions so, although promising, the Minoux's cannot be applied to help solve it.

2.6.4 Extensions of the TCFA Problem

In [McGi94] the authors compare the performance of Tabu Search (TS) and Simulated Annealing (SA) based procedures for solving TCFA problems with offset-linear link cost functions. The TS procedure starts with a complete network, while the SA procedure starts with a random topology. The TS procedure uses the accelerated greedy link elimination algorithm from [Mino89], using the tabu list to force the greedy search out of local minima. The SA procedure generates new solutions by selecting a link at random. If the link is present in the network it is removed, if it is not present it is added. The results presented for the design of six 20 node networks show little difference in the cost of the solutions found by the two procedures. However, the SA procedure consistently takes up to three times longer than the TS procedure. We do not consider either of these procedures further, as the SA procedure requires an excessively large number of solution cost evaluations, and as the TS procedure is based on the "flawed" accelerated greedy link elimination algorithm from [Mino89].

Kamimura and Nishino [Kami91] examine the Capacity Flow Assignment (CFA) problem in the context of an elementary network of one tandem switch and a number of local switches. They assume continuous link capacities and a fixed-charge link cost function. Although the authors discuss general concave link cost functions they do not use them. Thus Kamimura and Nishino's approach is not applicable to the DSNDP, which specifies concave link cost functions.

A fast design algorithm for mesh or distributed circuit switched networks, called MENTOR, is presented in [Kers91]. The goal of the algorithm is not to find the optimal solution, but a good solution that can be used as a starting point for further optimisation. Alternatively this algorithm could be embedded inside a design loop for designing hierarchical networks. Link costs are assumed to be concave functions of distance, stepwise concave with capacity, and incur a fixed installation cost.

A variant of the MENTOR algorithm, called MENTour, is presented in [Cahn95]. The MENTour algorithm ensures that the network designs are 2-connected (i.e. minimum node degree of 2). This is useful when link utilisations are low, as the MENTOR algorithm tends to produce tree networks when this is the case.

Although the MENTOR and MENTour algorithms consider concave link cost functions, two features make them unsuitable for use in solving the Concave TCFA problem and hence the DSNDP. The first is that they are designed to work with circuit switched networks where delay is not an issue. This means that the algorithms have no way to enforce delay based performance constraints. The second, more significant problem is in the assumption that, given continuous link capacities, the cheapest solution is to create a meshed network. With a cost function that is concave with respect to capacity and linear with respect to distance this is not so (see for example [Fran72], [Klei76], [Gerl77], [Gers90] and [Dutt92]) in which the best network designs are more tree like than meshed). A simple illustration that tree networks are cheaper than meshed networks when the link cost functions are concave with respect to capacity (f) and linear with respect to distance (d) is shown in Figure 2.8.

In [Dutt92] Dutta and Lim examine a multiperiod TCFA network design problem. An initial network topology and traffic demand matrix is assumed given. In each design period, the traffic requirements increase, through either a increase in traffic between each node pair, or the addition of nodes. The object is to determine where and when capacity should be installed to minimise the cost while satisfying Kleinrock's [Klei76] maximum packet queuing delay quality of service constraints. The authors employ a Lagrangian relaxation based approach, similar to that in [Gavi89], to solve the formulated multiperiod design problem. Like Gavish *et al.* Dutta assumes that the number of possible link capacities is small and solves a pure inte-

The length of a link = d

The traffic flow, or load, on a link = f

The cost of a link = $d \times f^{0.4}$.

The cost of a network, C, is the sum of the link costs.

C = 1 + 1 + 2 = 4

(a) Mesh

C = 1.3195 + 1.3195 = 2.639

(b) Tree

Figure 2.8 An illustration that tree networks are cheaper than meshed network when the link cost functions are concave with respect to capacity an linear with respect to distance.

ger programming problem via implicit enumeration to determine the capacity of each link. Like the exhaustive search employed by Gavish *et al.*, the execution time of an implicit enumeration of possible link capacities limits the usefulness of Dutta's TCFA algorithm when the number of possible link capacities is large.

Butto [Butt96] incorporates reliability constraints into the TCFA problem. Butto formulates the TCFA problem as a MINLP and constrains the network design such that there are at least two node disjoint paths between O-D pairs, and that all paths are either one or two hops. Node degree is also limited. Link costs are assumed to be linear with respect to capacity, which is assumed to be assigned in discrete increments. This produces a stepwise-linear link cost function, with the addition of a fixed installation cost. The design heuristic proposed by the author relaxes the integer constraints to obtain a convex NLP. Link capacities are then quantised using a branch-and-bound procedure. While the inclusion of network reliability constraints into this problem makes it more like the DSNDP, where they must also be considered, the linear cost functions result in a convex problem which is relatively easily solved. The concave cost functions in the DSNDP produce a concave problem which is not so easily solved.

2.6.5 Conclusions

Of the procedures available in the literature for solving TCFA, or related, problems, only Kleinrock and Gerla's CBE procedure, Gavish *et al.*'s MILP based procedure, and Gersht and Weihmayer's link elimination procedure are designed to cope with the general concave link cost functions that occur in the DSNDP. However, as discussed above, each of the existing procedures has short comings when applied to designing the network topology inherent in the DSNDP. Gavish *et al.*'s MILP based procedure is immediately disadvantaged by the large number of possible link capacities in an ATM based network. Also the complexity of Gersht and Weihmayer's link elimination procedure when the number of possible links in the network is large limits its usefulness. Finally, although Kleinrock and Gerla's CBE procedure should cope with concave link cost functions nothing to verify this has appeared in the literature to date.

Another important observation is that the majority of previous work on network design example problems have been generated by assuming uniform traffic requirements between all node pairs. This assumption produces well balanced network designs which may not be appropriate in practice. This also means that existing algorithms may have been (inadvertently) biased towards balanced network solutions. This issue is pointed out in [Dutt92] where the author uses random O-D traffic requirements in an attempt to overcome the problem. Although better than assuming uniform O-D traffic requirements Dutta's method is still not realistic. To generate realistic O-D pair traffic requirements for each network we have assumed that the level of interaction between nodes is proportional to the user population located at each node and inversely proportional to the distance between them. A procedure for generating traffic requirements based on this assumption is described in Chapter 6.

In Chapter 4 we examine and compare the performance of both Kleinrock and Gerla's CBE procedure and Gersht and Weihmayer's link elimination procedure in detail. Our results show that the CBE procedure does not perform nearly as well as the link elimination procedure, but as expected, the link elimination procedure is computationally very expensive. In order to design a network topology given a set

of server locations and the assignment of user nodes to them (i.e. an O-D traffic requirements matrix) we have developed a *Concave Link Elimination (CLE)* procedure, based on Gersht and Weihmayer's procedure. The CLE procedure is presented in detail in Chapter 4.

2.7 Conclusions

The DSNDP includes aspects from a variety of network design problems, including: facility, *p*-median, and hub location problems, along with centralised and backbone network design problems. While each type of design problem examined is similar to the DSNDP in some way, each is also different in some way. This means that the existing solution procedures for each of these types of problem cannot provide us with a complete solution to the DSNDP. Despite this, some of the solution procedures discussed in this chapter are of use in obtaining a complete solution to the design of a DSN. For example, in Section 2.6 we showed that once the location of servers and the assignment of user nodes to them is established, an external traffic requirements matrix can be generated. The problem then becomes a variation of a backbone network topology design problem.

Facility location problems are similar to the DSNDP in that the objective is to locate servers (i.e. facilities) and assign clients (i.e. demand nodes) to them in the most cost effective way. The traffic (or product) demand is known but the traffic volume from each server has to be determined. However, facility location problems are dissimilar to the DSNDP in that the topology of the network connecting clients to their server is not considered (clients are assumed to be directly connected to facilities). Unlike the DSNDP, facilities do not interact in traditional facility location problems. Finally, the cost functions assumed in most facility location problems are linear in contrast to the concave cost functions of the DSNDP. Hence, existing procedures for solving facility location problems are not appropriate for solving the DSNDP firstly because they do not include network topology design or interaction between facilities, and secondly because they rely on the linear form of the cost functions.

p-median location problems are similar to the DSNDP in that, like facility location problems, they attempt to determine the location of the median's (i.e. servers) and

the assignment of client nodes to them. In addition, the p-median problems add the constraint that nodes are connected via a network of links. However, p-median problems differ from the DSNDP in that, as in the facility location problem, the median's do not interact. In addition, the objective of the p-median problem is to minimise the sum of the link distance between each client and its associated median. In contrast, in a DSN, the cost of connecting a client node to its associated server depends not only on the link distance but also on the traffic volume on the links involved. This in turns depends on the traffic from other clients and how it is routed. To conclude, existing procedures for solving p-median problems are not appropriate for solving the DSNDP because in the first instance the number of median's to be located is given. Thus the solution procedures themselves are not designed to determine the optimal number of medians. Secondly, existing solution procedures generally enumerate a large portion of the potential solution space of the given p-median problem. As discussed earlier, this approach is not suitable for solving the DSNDP due to the expense of evaluating each potential DSNDP solution.

Hub location problems are similar to the DSNDP in that the objective is to locate a number of hubs (servers) that interact with one another via a backbone network to allow client nodes to communicate with each other. However, hub location problems are dissimilar to the DSNDP in a number of ways. Firstly, node interaction is assumed to be between client nodes via their respective hubs and the backbone network. In a DSN the interaction is assumed to be between clients and their associated server, and between servers, rather than only between clients. Secondly, in a hub location problem, clients are assumed to be directly connected to hubs and hubs to one another. Thus, the topology of the network is not included in the problem. Thirdly, the (offset-)linear link cost functions assumed in hub location problems cannot capture the economies of scale captured by the concave link cost functions employed in the DSNDP. Moreover, as with the p-median problem, the number of hubs to be located is usually supplied rather than being an output of the design process. Nor is the capacity of hubs considered. Lastly, most of the existing solution procedures for hub location problems are not appropriate for determining the number of servers and assignment of clients to them in a DSNDP. This is because they rely on either the linear cost functions to formulate MILP's that can be solved as part of the solution procedure, or on evaluating a large number of potential solutions to the

problem. However, the clustering based procedures of [OKe92] and [Diri77] show potential for the DSNDP, as they cluster nodes together such that intra-cluster interaction is maximised while minimising inter-cluster interaction. In addition, they do not rely on either the form of the cost functions, or evaluating a large number of potential solutions. Node clustering based procedures appear again in the next chapter and we develop our own node clustering based procedure in Chapter 5.

Centralised network design problems are similar to the DSNDP in that, like p-median problems, they work out how to connect clients to a server via a network of links. Centralised network design problems go further than the p-median problem by imposing constraints on the traffic on each link between the clients and server. However, centralised network design problems are dissimilar to the DSNDP in that they either assume that there is only one server to which all clients are to be connected, or that the servers are not connected to one another in any way. If multiple servers are involved, then the assignment of clients to them is an input to the process, rather than an output. Also, none of the centralised network design problems include performance or reliability constraints on the solution. The inclusion of these constraints, as in the DSNDP, complicates the problems and requires the development of new solution procedures. Indeed, the inclusion of performance and/or reliability constraints is discussed as an area for future work in [Gavi91]. To our knowledge no further work in this area has appeared in the literature. Solution procedures for centralised network design problems are not appropriate for solving the DSNDP because even when link cost is dependant on capacity it is assumed to be (offset)-linear with a small number of possible link capacities. As discussed previously neither the assumption of (offset)-linear cost functions, or of only a few possible link capacities, is realistic in the ATM based environments likely to accompany DSN's.

Backbone network design problems are similar to the DSNDP in that they consider the interconnection of a number of nodes via a network of links, and they take the traffic on the links and the performance of the network as a whole into account. Some researchers, including [Klei76], [Gerl77], [Mino89], [Gers90] and others, even assume that link cost functions are concave with respect to link capacity, to capture the effect of economies of scale in link capacity pricing. However, backbone network design problems are dissimilar to the DSNDP in that they ignore the access

portion of the network completely. Of the procedures available in the literature for solving TCFA, or related, problems, only Kleinrock and Gerla's CBE procedure, Gavish *et al.*'s MILP based procedure, and Gersht and Weihmayer's link elimination procedure are designed to cope with the general concave link cost functions that occur in the DSNDP. However, as discussed above, each of the existing procedures has short comings when applied to designing the network topology inherent in the DSNDP. Gavish *et al.*'s MILP based procedure is immediately disadvantaged by the large number of possible link capacities in an ATM based network. The complexity of Gersht and Weihmayer's link elimination procedure when the number of possible links in the network is large limits its usefulness. Finally, although Kleinrock and Gerla's CBE procedure should cope with concave link cost functions nothing to verify this has appeared the literature to date.

In Chapter 4 we examine and compare the performance of both Kleinrock and Gerla's CBE procedure and Gersht and Weihmayer's link elimination procedure in detail. Our results show that the CBE procedure does not perform nearly as well as the link elimination procedure, but as expected the link elimination procedure is computationally very expensive. In order to design a network topology given a set of server locations and the assignment of user nodes to them (i.e. an O-D traffic requirements matrix) we have developed a *Concave Link Elimination (CLE)* procedure, based on Gersht and Weihmayer's procedure. The CLE procedure is presented in detail in Chapter 4.

Even more useful than the network design procedures discussed in this chapter are solution approaches proposed for solving problems that combine two or more of the problems discussed in this chapter. In the next chapter, we present a mathematical formulation of the DSNDP, which illustrates how the DSNDP combines the problems from this chapter. Following this is a discussion of previous work on similar combined problems, and how we approach the solution of the DSNDP in light of previous work.

3. Problem Formulation and Solution Approaches

There is more than one way to skin a cat.

- Unknown.

3.1 Introduction

In this chapter we present the DSNDP as a mathematical optimisation problem. The mathematical formulation of the problem clarifies how the DSNDP combines aspects of a number of the individual network design problems reviewed in Chapter 2. We then review previous work on network design problems that, like the DSNDP, combine aspects of two or more of the individual problems discussed in Chapter 2. Based on the formulation of the DSNDP and our review of previous work and the solution approaches employed, we also discuss how the DSNDP may be decomposed and solved. This chapter concludes with a summary of the deficiencies in previous work, and developments in the state of the art of DSN design presented in this dissertation.

3.2 Mathematical Programming Formulation

In this section we formulate the DSNDP as a multicommodity flow problem. A DSN consists of populations of clients and server locations with links connecting them. Client populations, server locations and transmission links are mapped to the nodes and edges, respectively, of a directed graph $G = (N, L)$ (where N is the set of network nodes, i.e. client populations and server locations, while L is the set of

links). Capacity is assigned to each link to allow it to carry traffic in either direction (a link with zero capacity is effectively removed from the network).

3.2.1 Assumptions

Following Kleinrock [Klei76] and others in the area of network design we assume:

1. Length $i \rightarrow j$ = length $j \rightarrow i$.
2. Poisson arrival of external traffic at each client node.
3. Exponential packet length distribution.
4. Infinite buffers at nodes.
5. Error-free links.
6. Link propagation delay and nodal processing delays are ignored.

Assumptions 2 through 5, plus Kleinrock's *independence assumption* [Klei76], allow the network to be modelled as a system of M/M/1 queues. This allows us to derive expressions for the average packet queuing delay in the network. Assumptions 1 and 6 simplify the mathematical programming formulation of the problem, but may be relaxed by heuristic design procedures.

As is typical in the area of network design we assume that all traffic follows static, non-bifurcated, shortest path routes. To simplify the design problem we also assume that network reliability constraints are satisfied by the underlying transmission system (i.e. we do not impose any connectivity constraints on the design).

In addition, we assume that client populations (i.e. nodes) are assigned to only one server in the network. This means that all traffic generated at a given client node will be delivered to a single server in the network. In practice it may be desirable to assign each client node to a single primary server and a number of backup servers (see for example [Pirk88]). However, in many client-server applications, such as World Wide Web (WWW) proxy services, the Domain Name System (DNS), Object Request Brokering (ORB) services, clients are assigned, primarily, to only one server. We assume that the traffic delivered to a server by its associated client nodes will produce some volume of return traffic from the server to the clients. We

assume that the return traffic will follow the same path (in reverse) as the forward traffic. This return traffic is incorporated into the mathematical formulation of the problem by scaling the volume of traffic generated at each client node appropriately.

To capture the effects of inter-server communication on the network, we assume that some percentage, δ, of the traffic received by any given server is distributed uniformly amongst the other servers in the network. The volume of inter-server traffic, I_{kj}, transmitted from server k to j is:

$$I_{kj} = \begin{cases} \dfrac{\delta L_k}{|S|-1} & , k \neq j \\ 0 & , k = j \end{cases} \quad , \forall k, j \in S \tag{3.1}$$

where L_k is the total volume of user traffic offered to server k, while S is the set of servers in the network ($|S|$ denotes the size of set S).

3.2.2 Input Variables

The following variables are supplied as inputs to the design process by the network designer.

N = set of all nodes.
P = set of potential server locations ($P \subseteq N$).
R = set of allowed server capacities.
Q = set of allowed link capacities.
L = set of link pairs (i.e. one link in each direction between nodes) allowed in the network.
l_{ij} = the length of the link, $(i,j) \in L$, connecting nodes i and j.
γ_i = traffic supply at node i destined for its server.
α_i, β_i = link and server cost function parameters.
C_{ij} = maximum capacity of link, $(i,j) \in L$, connecting nodes i and j.
C_k = maximum capacity of server located at node $k \in P$.
T_{max} = a limit on the average packet queuing delay.
$\zeta(x)$ = is a function of x which equals 1 if $x > 0$ and 0 otherwise.

3.2.3 Decision (or Output) Variables

The values of the following variables are produced as outputs of the design process.

f_{ij} = directed client-server traffic flow from node i to j on link $(i,j) \in L$.

c_{ij} = undirected capacity[1] of link $(i,j) \in L$ connecting nodes i and j (there is no significance in the order of the subscripts, i.e. $c_{ij} = c_{ji}$).

f_{kij} = directed inter-server traffic flow generated by the server at node $k \in P$ flowing from node i to j on link $(i,j) \in L$.

x_k = the total client-server traffic (not including inter-server traffic) on the server at node $k \in P$.

w_k = the capacity of server at node k. $w_k \in R$, $\forall k \in P$, $w_k = 0$ indicates that there is no server at node k.

For ease of expression the total undirected traffic (both client-server and inter-server) on a link is denoted F_{ij} where

$$F_{ij} = f_{ij} + f_{ji} + \sum_{k \in P} (f_{kij} + f_{kji}), \forall (i,j) \in L \tag{3.2}$$

3.2.4 Network Component Cost Functions

The cost of a network (the objective function in this context) consists of the cost of the links and servers in the network.

The assignment of undirected capacity, c_{ij}, to a link $(i,j) \in L$ incurs both a *termination* cost (e.g., costs associated with the switch interfaces on either end of the link) proportional to the capacity, c_{ij},

$$\text{termcost}(c_{ij}) = \beta_1 c_{ij}^{\alpha_1} + \beta_2 c_{ij}^{\alpha_2} \tag{3.3}$$

and a *line* cost proportional to both the capacity, c_{ij}, and length, l_{ij}, of link $(i,j) \in L$

$$\text{linecost}(c_{ij}, l_{ij}) = \left(\beta_3 c_{ij}^{\alpha_3} + \beta_4 c_{ij}^{\alpha_4}\right) l_{ij} \tag{3.4}$$

The cost of a server, $k \in P$, is dependent only on its capacity, c_k.

$$\text{cost}(c_k) = \beta_5 c_k^{\alpha_5} + \beta_6 c_k^{\alpha_6} \tag{3.5}$$

1. The undirected capacity of a link is the sum of the capacities required in each direction of the link, which may be split in each direction arbitrarily.

where $\beta_{i = 1, \ldots, 6}$ and $0 \leq \alpha_{i = 1, \ldots, 6} \leq 1$ are coefficients supplied by the network designer to model link and server costs. We use a *double power-law* link cost function (Equations (3.3), (3.4), and (3.5)) to model the economies of scale that exist in the pricing of communication network components. The double power-law cost function, introduced in [Stac97b], is an extension of the *power-law* cost function (Equation (3.6)) used in [Klei76], [Gerl77] and [Dutt92].

$$\text{cost}(c_{ij}) = \beta c_{ij}^{\alpha} + K \tag{3.6}$$

The advantage of replacing the constant offset, K, with a second *power-law* term is twofold. Firstly, the cost function becomes smooth, the constant offset, K, in Equation (3.6) means that the surface of the cost function includes step changes as links are added or removed. Using a smooth cost function allows the use of gradient based optimisation methods in the design of the network. Secondly, with appropriate parameters the double power-law function can be made to match a wider variety of link cost points (including a single power-law function with a constant offset).

We assume that the cost associated with link end points (i.e. switching nodes) is included within the cost of a link (this is also done in [Cahn91] [Klei76] [Gerl77] [Kers91] [McGi94] [Kami91] [Gers90] and [Gavi89], amongst others).

In addition, we assume that link and server capacity is available in discrete increments. In ATM networks link capacity is provided by virtual paths in the underlying network. Virtual paths can be assigned in any capacity (up to the limit of the underlying transmission system) so link capacity increments would be small. In non-ATM networks link capacity increments may be large. In either environment the "capacity" of servers will be determined by the memory, storage and processing hardware available to them. The server capacity increment represents the stepwise increases in the capacity of a server due to the addition of resources.

3.2.5 Mixed Integer Non-Linear Programming Formulation

Using the notation and assumptions above, the DSNDP can be formulated as a MINLP as follows:

$$\min\ Z(c,w) = \sum_{\{i,j\}\in L}\left\{\beta_1 c_{ij}^{\alpha_1} + \beta_2 c_{ij}^{\alpha_2} + \left(\beta_3 c_{ij}^{\alpha_3} + \beta_4 c_{ij}^{\alpha_4}\right) l_{ij}\right\} + \sum_{k\in P}\left\{\beta_5 w_k^{\alpha_5} + \beta_6 w_k^{\alpha_6}\right\} \tag{3.7}$$

Subject to:

$$\gamma_i + \sum_{\{j\ni(i,j)\in L\}} f_{ji} - \sum_{\{j\ni(j,i)\in L\}} f_{ij} = \left\{\begin{array}{l} x_i \ \ i\in P \\ 0 \ \ i\notin P \end{array}\right\}, \forall i\in N \tag{3.8}$$

$$\sum_{\{j\ni(i,j)\in L\}} f_{kji} - \sum_{\{j\ni(i,j)\in L\}} f_{kij} = \left\{\begin{array}{ll} -\delta x_k & i=k \\ 0 & i\notin P \\ \dfrac{\delta x_k \zeta(x_j)}{\left(\sum_{i\in P}\zeta(x_i)\right)-1} & i\neq k, i\in P \end{array}\right\} \forall i\in N, k\in P \tag{3.9}$$

$$\left(\frac{1}{(1+\delta)\sum_{i\subset N}\gamma_i}\right)\sum_{(i,j)\in L}\left(\frac{F_{ij}}{c_{ij}-F_{ij}}\right) \le T_{max}, \forall (i,j)\in L \tag{3.10}$$

$$\lceil F_{ij}\rceil_Q \le c_{ij}, \forall (i,j)\in L \tag{3.11}$$

$$c_{ij} \le C_{ij}, \forall (i,j)\in L \tag{3.12}$$

$$\lceil x_k(1+\delta)\rceil_R \le w_k, \forall k\in P \tag{3.13}$$

$$w_k \le C_k, \forall k\in P \tag{3.14}$$

$$f_{ij} \ge 0, \forall (i,j)\in L \tag{3.15}$$

$$f_{kij} \ge 0, \forall (i,j)\in L, k\in P \tag{3.16}$$

$$x_k \ge 0, \forall k\in P \tag{3.17}$$

$$w_k \ge 0, \forall k\in P \tag{3.18}$$

$$w_k \in R, \forall k\in P \tag{3.19}$$

$$c_{ij} \ge 0, \forall (i,j)\in L \tag{3.20}$$

$$c_{ij} \in Q, \forall (i,j)\in L \tag{3.21}$$

Constraints (3.8) and (3.9) ensure the conservation of client to server and inter-server traffic flow respectively. For client to server traffic Constraint (3.8), states that for all nodes i, the volume of traffic generated at node i (γ_i), plus the outgoing traffic (i.e. from i to each adjacent node j) less incoming traffic (i.e. from each node j to i) must equal either:

- the volume of client to server traffic consumed by the server located at node i, if node i is a potential server location;
- or, zero (if node i is not a potential server location).

For inter-server traffic (Constraint (3.9)) the inter-server traffic generated by each server is treated as a separate commodity. So, for each commodity (potential server location), k, for all nodes i, outgoing traffic (i.e. from i to each adjacent node j) less incoming traffic (i.e. from each node j to i) must equal either:

- $-\delta x_k$, the negative of the volume of inter-server traffic generated by server k, if node i is the location of server k (if $x_k = 0$ and there is no server at node i, i.e there is no traffic is generated);
- $(\delta x_k \zeta(x_j))/((\sum_{i \in P} \zeta(x_i)) - 1)$, if node i is a potential server location other than k, and the server at node i is active (i.e. $\zeta(x_j) = 1$), and node i is a destination of inter-server traffic generated by server k. The volume of traffic from server k, δx_k , is divided amongst the active servers in the network. The number of active servers (excluding k) is given by $(\sum_{i \in P} \zeta(x_i)) - 1$;
- or, zero (i.e. node i is not a potential server location).

Constraint (3.10) ensures that the average packet queuing delay is less than the specified value T_{max} (see [Klei76], Equation 5.19). Constraint (3.11) ensures that the capacity assigned to a link, c_{ij}, is greater than or equal to the total traffic on it. $\lceil x \rceil_Q$ denotes the smallest allowable capacity, from the set Q, larger than x. Constraint (3.12) ensures that the maximum link capacity, C_{ij}, is not exceeded. Constraint (3.13) ensures that if a server is opened at a potential server location it is of the appropriate capacity. This is achieved by combining the client-server traffic load, x_k, and inter-server traffic load, δx_k , on a server. Constraint (3.14) ensures

that the maximum capacity of servers, C_k, is not exceeded. Constraints (3.19) and (3.21), ensure link and server capacities are selected from the predetermined set of options.

Constraints (3.15), (3.16), (3.17), (3.18) and (3.20) ensure the validity of the decision variables.

The discrete sets of allowable link and server capacities, Q and R, make this problem a MINLP optimisation problem.

3.3 Combined Network Design Problems

In Chapter 2 we examined specific network design problems and the approaches taken to solve them. The DSNDP includes a number of those specific design problems, however none of these specific problems completely addresses the DSNDP. In this section we examine other design problems created by combining two or more of the problems described in Chapter 2, and the solution approaches taken to solve them.

3.3.1 Facility Location and Centralised Network Design

In this section we examine solution approaches to problems that combine aspects of facility location (Section 2.2), p-median (Section 2.3), and centralised network design (Section 2.5) problems. These types of problems are related to the DSNDP as they involve the location of facilities (i.e. servers) and the design of the link topology to connect client nodes to them.

3.3.1.1 Node Clustering Based Approaches

In [McGr77] the objective is to connect user nodes to one of several central sites via concentrators, referred to as Generic Access Facilities (GAF's). Nodes are connected to GAF's via multidrop lines and GAF's to the central site(s) via point-to-point links. The number of nodes that can appear on a multidrop line is restricted, as is the total number of nodes that each GAF can support. A gravity based clustering procedure is employed to identify groups of nodes to be attached to the same multipoint line.

The gravity based clustering algorithm operates by merging pairs of nodes together and replacing them with a single node located at their Centre of Mass (CoM) (the mass of a node is the number of nodes that is represents). The algorithm starts by merging pairs of nodes that are closest together. If merging any pair of nodes would violate the constraint on the multidrop line capacity, that particular merger is removed from future consideration. The clustering algorithm terminates when no two nodes can be merged without violating any constraints.

Once the original set of nodes have been replaced by a reduced set of CoM nodes by the clustering algorithm, an ADD based algorithm is used to determine the number of GAF's required and assign the remaining CoM nodes to them. Initially, all CoM nodes are connected to the central site via point-to-point lines. The ADD algorithm operates by iteratively designating each CoM as a GAF. The benefit of connecting every other CoM node to the GAF rather than the central site is evaluated. The total benefit of placing a GAF at a CoM node is determined by the set of CoM nodes which benefit most from being connected to the GAF, up to the capacity of the GAF. Having evaluated all possible CoM nodes as potential GAF sites, the one with the maximum benefit is designated as a GAF site. The CoM node (and the other CoM nodes assigned to it) are then removed from further consideration. The process continues until the addition of a GAF provides no benefit. At this point the remaining CoM nodes are assigned to be connected directly to the central site.

Once the CoM nodes are partitioned, the k ($k = 3$, in [McGr77]) actual nodes closest to each CoM selected as a GAF site are evaluated as locations for the GAF site for that partition. The k potential GAF sites in each partition are evaluated using the same benefit measure as in the partitioning phase. An unspecified line layout algorithm is then employed to solve the CMST problem of connecting the user nodes to the GAF in each partition.

The design procedure from [McGr77] is extended in [Schn82] to allow the use of multiple levels of GAF's and multiple GAF capacities. In contrast to [McGr77] terminals are connected directly to GAF's in [Schn82], rather than via multidrop lines.

A shortcoming of gravity based clustering schemes is evident in both [McGr77] and [Schn82]. To prevent the algorithm from continuing to merge pairs of nodes until everything collapses into one large cluster a limit must be placed on the maximum

size of a cluster. In [McGr77] the maximum capacities of a multidrop line and a GAF are used. Multidrop lines are not used in [Schn82] and GAF's can take up a variety of capacities. The size of clusters are limited, in [Schn82], by specifying that two nodes must be closer than a given distance apart to be considered for merging. While the limits in [McGr77] are clearly justifiable, they limit the procedure to considering only fixed capacity GAF's. The maximum distance limit in [Schn82] appears arbitrary and the value chosen will have a significant effect on the solutions produced. Both approaches rely for success on decisions made by the designer. In a complex problem, such as DSNDP's are likely to be, it would become almost impossible for the designer to chose ideal capacities or distances without resorting to an iterative search.

Another example of using a clustering based approach to solve a similar problem to that in [McGr77] is presented in [Dysa78]. The authors employ a simple clustering algorithm to determine the number and location of concentrators in the network. The clustering algorithm in [Dysa78] operates by first calculating the K-nearest-neighbour node sets of each node. The weighted mean number of K-nearest-neighbour sets that each node appears in is then calculated. All nodes that appear in more K-nearest-neighbour sets than the mean are designated as concentrator locations (up to a maximum of $N/2$, where N is the number of nodes in the network). To complete the clustering process, terminals are assigned to their nearest concentrator. The terminals are further partitioned by applying a multidrop line layout algorithm (see reference [10] in [Dysa78]) to solve the centralised network design sub-problems of connecting terminals to their concentrator. The set of nodes on each multidrop line attached to each concentrator are considered as a single node (referred to as a *super-node*) in the next phase of the design in which concentrators are dropped and their associated super-nodes reassigned in an effort to reduce the cost of the design.

One of the problems with a median finding algorithm, such as the K-nearest-neighbour algorithm employed in [Dysa78], is that nodes that are close together will often have the same (or very similar) K-nearest-neighbour sets. This means that the algorithm will often over estimate the number of medians, and place some of them very close together. A DROP algorithm, as employed in the second phase of [Dysa78], can be employed to eliminate spurious medians.

Another variation on combining a concentrator location problem with a centralised network design problem is examined by Pirkul and Nagarajan in [Pirk92]. In this problem, terminals are connected to concentrators via direct links, creating a star topology. However, rather than concentrators also being connected to the central site via direct links, they may be connected to one another on backbone paths that terminate at the central site. The algorithm operates in two phases. The first phase uses a *sweep* algorithm to divide the terminals geographically into sectors centred on the central site. The number and size of the sectors are controlled by a maximum allowable sector angle and maximum total traffic allowed within a sector. Once the terminals have been divided into sectors, the problem of determining a path from the most remote node in each sector to the central site is formulated as an NP-complete Integer Program (IP). All nodes in the paths (except the start and central node) are designated as concentrators. The IP is solved using a Lagrangian relaxation based heuristic with sub-gradient optimisation being used to update the Lagrangian multipliers. An illustration of the type of topology produced by the algorithm is shown in Figure 3.1 (adapted from [Pirk92]).

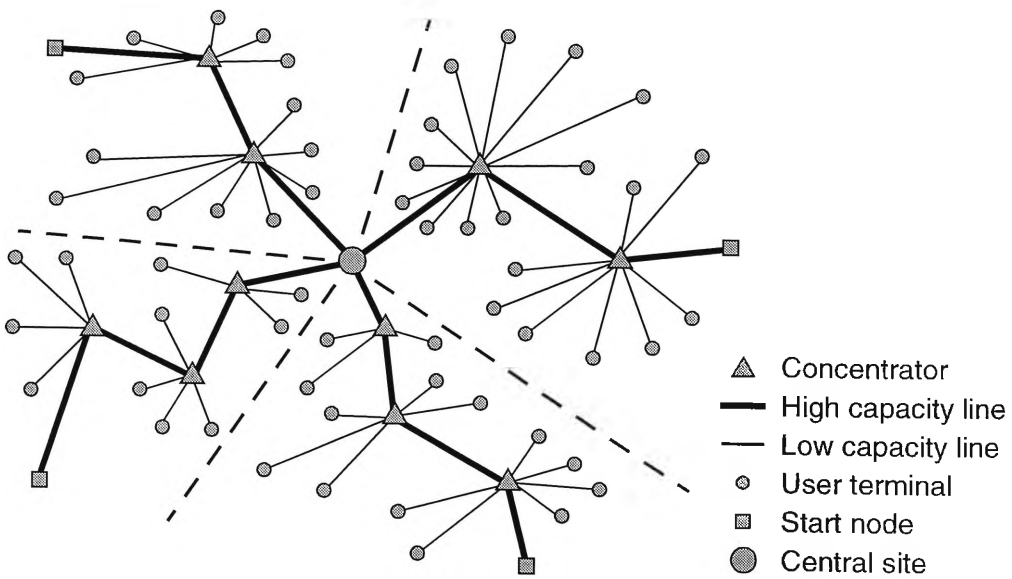


Figure 3.1 Star-backbone path topology for centralised network design from [Pirk92].

The sweep employed algorithm in [Pirk92] assumes that the central site is reasonably near the geographic centre of the user terminals. If it is not, the sectors will become very narrow and the backbone paths almost parallel with one another - a design which would clearly be sub-optimal. In addition, the sweep algorithm deter-

mines the boundary of sectors based only on the angle or traffic volume within the sector. The actual location of nodes with respect to one another is not taken into account. This means that a sector boundary could easily split a group of nodes which should be kept together. The second phase design algorithm (which appears in [Curr91] as well) determines the backbone path of concentrators in each sector, and assigns user terminals to concentrators. It is limited in that the cost of concentrators and links is considered to be independent of their capacity. Furthermore, the IP does not consider any quality of service or performance constraints. In contrast, the DSNDP includes concave link costs, delay and (optionally) network reliability constraints.

3.3.1.2 Mathematical Programming and Lagrangian Relaxation Based Approach

In [Pirk96] Pirkul again considers a centralised network design problem that combines both a concentrator location and link topology design problem. In [Pirk96] the objective is to connect user terminals to concentrators via tree topologies, and concentrators to the central site via a star topology. In contrast to [Pirk92] and [Curr91], the capacity of concentrators and links is included in the design problem. However, link capacities are fixed, unlike in the DSNDP. The design problem is formulated as a MILP single commodity flow problem. A Lagrangian relaxation of the full problem is formed, which is split into three subproblems. Unfortunately, while mathematical formulations of the subproblems are provided, the reader is left without any explanation of them. Also a heuristic is described that generates a feasible solution from the solutions of each Lagrangian subproblem.

3.3.1.3 Conclusions

The most promising design algorithms that appear in this area are those based on node clustering algorithms ([Diri77], [Dysa78], [Schn82]). They illustrate the ability of node clustering algorithms to identify groups of nodes that should be assigned to the same facility in complex facility location problems. Importantly, they are able to do so without relying on specific characteristics of either the problem, or its formulation. In addition, they do so without evaluating large numbers of potential solutions. This is particularly important for a problem such as the DSNDP, where the

evaluation of each potential solution is computationally expensive. In Chapter 5 we develop a node clustering algorithm to solve the DSNDP.

3.3.2 Hub Location and Backbone Network Design

In this section we examine solution approaches to problems that combine aspects of both hub location (Section 2.4) and backbone network design (Section 2.6) problems. These types of problems are related to the DSNDP as it also involves the location of facilities (i.e. servers) interconnected via a backbone network.

As mentioned in Section 2.6, the design of a backbone network requires a complete O-D pair traffic requirements matrix as an input. A significant feature of both DSNDP's and combined hub and backbone network design problems is that a complete description of the flow of traffic between all nodes in the network is not available as an input. In the case of a DSN it is known that client traffic must travel to (and from) a server, and that there will be traffic flowing between servers. However, it is not known how many servers there are, where they are located, or which client traffic is to be delivered to which server and hence the flow of inter-server traffic. In the case of a combined hub and backbone network design problem the volume of traffic to be delivered between user nodes is known. However, this traffic cannot travel directly, and instead must travel via one or more backbone nodes (or switches). The number, location, and assignment of user nodes to switches, and hence the traffic between switches, are all unknown.

Progress can be made on both problems by observing that once the location of backbone nodes and the assignment of user nodes to them is known, a complete description of the traffic requirements between all pairs of nodes (user and backbone) can be generated for either problem. This means that both the DSNDP, and combined hub location and backbone network design problems, can be decomposed into two subproblems: (i) determining the number and location of backbone nodes along with the assignment of user nodes to them, and (ii) the design of the network topology connecting user to backbone nodes and backbone nodes to one another. Hence the solution procedures for the combined hub location and backbone network design problem may also be used to design DSN's, and vice versa.

In [Monm86] Monma and Sheng examine a design problem in which user nodes are directly connected to switches, which are in turn connected by a fully meshed network. User nodes may be either access facilities representing a collection of low-volume users, or individual high-volume users. Traffic is routed via minimum hop paths, with traffic distributed uniformly over equal minimum hop paths between O-D pairs. User nodes are assumed to want to home to either one switch, all switches, or some subset of switches. Links are assumed to be available in a variety of types, each with different Grades of Service (GoS) (Bit Error Rate (BER)), virtual-circuit capacity, speed, and cost per mile. Switches have a fixed uniform capacity, but their cost has both a fixed and variable component. The variable component of the switch cost is dependent on the number and type of node processing units (NPU's) required within each switch (one NPU is required for each link type).

The design procedure starts by estimating the number of switches required, S, by assuming that both user nodes and traffic will be evenly distributed amongst switches and dividing the effective traffic by the effective capacity per switch. A clustering algorithm is employed to partition the user nodes into S clusters, each of which will be served by a single switch located at its centre of mass. Users nodes that have to be homed to all switches are automatically members of all clusters. User nodes with a high volume of traffic are initially assigned to all clusters. Remaining nodes are assigned (in order of descending throughput) to the cluster that maximises a "goodness of fit" measure for that node-cluster pair. Once all nodes have been assigned, the clustering is perturbed to ensure that all nodes are assigned correctly. End-to-end blocking probability and delay measures are used to assign facility types to links and load switches. The resulting network design is analysed using Ward Whitt's Queuing Network Analyser (QNA) (with retransmissions) (see references in [Monm86]). Any bottlenecks in the network are identified and the design module reinvoked to add capacity to the areas identified.

The design procedure in [Monm86] has two main limitations. The first is the use of fixed capacity switches. The second is the assumption of a fully meshed network. The latter means that no link topology design is required. These limitations greatly simplify the problem and allow the use of a relatively complex analysis procedure (QNA) to refine link capacity assignments. However, [Monm86] shows how an

analysis package, such as QNA, can be employed to refine the final solution of any design procedure.

An interactive design tool based on [Monm86] is described in [Card89].

Cahn *et al.* [Cahn91] describe an interactive tool to aid in the design of communication networks. In general, user nodes are assumed to be connected to backbone nodes (switches) via local access network topologies (star or tree topologies), while switches are connected via a backbone network. The main focus of the paper is on the development of an interactive design tool that allows designer's knowledge to guide the design of the network. This focus motivates the use of design techniques that are sub-optimal but fast so that design alternatives can be presented to the user quickly.

The design of the network is split into three phases. The first phase determines the number and location of backbone nodes. This phase produces a complete traffic matrix for all nodes in the network. Once the first phase is complete the next two phases can proceed independently and in parallel. These two phases design the access and backbone portions of the network respectively.

The first phase of the design process, node selection, is performed using either of two different methods. The first is a threshold method in which a node is designated as a backbone node if the sum of its inbound and outbound traffic exceeds a user-specified threshold. A radius in terms of link cost is defined and any other node within that radius of the backbone node is designated an end node. End nodes are then assigned to their nearest backbone node. The threshold method works well when there are natural backbone nodes. If this is not so, then a median finding algorithm is used. The first phase tries to find a user-specified number of median nodes that minimise the cost of directly connecting user nodes to backbone nodes. The two procedures can be used together, with the threshold method being used to select nodes based on traffic and the median method being used to select backbone nodes to cover the remaining nodes.

The design of the backbone network is performed using the MENTOR algorithm from [Kers91] (also discussed in Section 2.6). The design of the access portions of

the network is then done using one of a variety of centralised network design algorithms (see references in [Cahn91]).

The items of most interest in [Cahn91] are the backbone node selection methods, described above (the limitations of the backbone network design algorithm employed, MENTOR, are discussed in Section 2.6). The threshold method is limited by the need for input from the designer to specify the traffic level that will determine if a node is selected as a backbone node. In effect, the number of backbone nodes will be equal to the total volume of traffic in the network divided by the user specified threshold level. However the designer could repeat the design process using a different threshold level to iteratively determine the best number of backbone nodes. The most significant problem with the threshold method is that it does not take into account the relative location of nodes. Some method is required to ensure that if two highly active nodes are close to one another they will not both be selected as backbone nodes. We investigate the effectiveness of such a procedure in Section 5.2.3. Median finding algorithms do not suffer the same problems as the threshold algorithm, and can also be employed iteratively, hence a median finding procedure is more applicable to the DSNDP. We apply a median finding procedure to the DSNDP in Section 5.2.3.

In [Gavi92a], Gavish assumes that user nodes are directly attached to switches, which are in turn connected by a backbone network, the topology of which is to be determined. Again, the approach is to first determine a set of backbone nodes (number and location) and the assignment of user nodes to them. This allows a complete traffic requirements matrix to be generated and the backbone network to be designed. The backbone network design procedure employed is presented in [Gavi86b], [Gavi89], [Gavi90], [Gavi92a], and was discussed in Section 2.6.

The entire design problem is formulated as a MINLP. A lower bounding problem is formulated by applying Lagrangian relaxation to the original problem. The relaxed problem is decomposed into a collection of smaller problems which can be solved to provide lower bounds on the optimal solutions to the original problem.

In addition, [Gavi92a] presents three heuristics for providing feasible solutions to the original problem. The first algorithm is a *greedy DROP heuristic*. It starts by designating all potential backbone nodes as backbone nodes. The individual

removal of each backbone node is evaluated and the node whose removal improves the solution cost most is dropped from consideration permanently. The second algorithm is called a *partial-enumeration heuristic*. This algorithm considers all possible combinations of 1 to k_{max} backbone nodes. The complexity of this algorithm is $O(B|I|^k)$, where I is the set of possible backbone node locations, and B the complexity of the TCFA problem associated with evaluating each combination of backbone nodes. Due to the complexity of the partial-enumeration algorithm, it is limited to small values of k_{max} (3 or 4 for most practical problems) [Gavi92a]. A third heuristic combines aspects of the previous two. It is a *greedy ADD-k heuristic* that starts without any backbone nodes and adds up to k_{max} backbone nodes in each iteration. The number of backbone nodes added in each iteration is determined by evaluating all possible combinations generated by adding from 1 to k_{max} backbone nodes to the current configuration. In all three heuristics, user nodes are assigned to their nearest backbone node.

In Section 9 of [Gavi92a] Gavish observes that the problem considered may be extended in a number of ways. For example, both link and backbone node switching capacities are assumed to be fixed. The design procedure of [Gavi92a] also excludes the design of the access portion of the network. In addition, no reliability constraints are placed on the network design.

Despite the simplifying assumptions made in [Gavi92a] the ADD-k heuristic offers a powerful method of determining the optimal number and location of backbone nodes. Its power comes from the fact that it is simple and does not rely on any constraining feature of the problem formulation, yet by looking up to k steps into the possible future of the search it is able to avoid small local minima. A slightly modified version of Gavish's ADD-k heuristic is adapted to aid in the design of DSN's in Section 5.3.2.

In [Zhan96] the design problem from [Gavi92a] is extended to include a variety of link and backbone node types (i.e. capacities). The extended problem is formulated as a non-linear program, and a lower bounding problem formed using linear relaxation. A greedy heuristic is developed that starts with a shortest path solution. Although not stated in [Zhan96], the assumption seems to be that all nodes may be backbone nodes and any nodes required to perform a switching role in the shortest-

path solution are designated as such. The heuristic proceeds by attempting to reroute traffic from the most expensive link and the most expensive switch until no further improvement can be found. The heuristic agrees well with the lower bound obtained by the Lagrangian relaxation from [Gavi92a] when there is a lot of traffic, as the quantisation of the link capacities to discrete values is not significant. However, when traffic is sparse, the quantisation error is significant. Furthermore, the heuristic algorithm presented in [Zhan96] relies on the assumption that traffic is required to flow between user nodes. Thus it is unsuitable for use in the design of a DSN.

In [Mukh93a], [Mukh93b] and [Saha95], Saha *et al.* develop a clustering algorithm to aid in the design of what they refer to as a two level communications network in the presence of node and link failures. Their design of a two level network is similar to a DSN, in that there is a gateway node in each cluster through which all inter-cluster traffic flows. Unfortunately, the authors appear to contradict themselves in [Saha95], [Mukh93a] and [Mukh93b]. When describing the proposed structure of the network (p. 379 of [Saha95]) the authors say that clusters are all the same size., and a method to determine the optimal size of clusters in each level of the hierarchy is presented on p. 381 of [Saha95] (and in [Mukh93b]). In contrast, the clustering algorithm presented on the same page (and in [Mukh93a]) produces clusters of varying sizes, as shown by the results in [Saha95] and [Mukh93a]. In addition, the authors state that each node has an associated "traffic demand," c_i (p. 380). Yet they do not explain where that traffic is to be delivered to, or supplied from. Their clustering algorithm groups nodes together based on the geographic proximity. The node in each cluster closest to the centre of the network is chosen to act as the gateway for that cluster. The design of the intra and inter-cluster link topologies is not discussed by the authors.

Despite the lack of clarity in the presentation of their node clustering algorithm, it offers a powerful way to identify promising configurations of backbone nodes and assignment of user nodes to them. The power of the clustering algorithm comes from the use of a "pull" or "gravity" style function to identify relationships between nodes. In addition, the algorithm does not rely on any arbitrary user input to guide it to a solution. In [Saha95], [Mukh93a] and [Mukh93b] the pull function is based purely on the geographic distance between nodes. In Section 5.2, we show how a

node clustering algorithm based on Saha *et al.*'s work can be augmented to take factors such as the link distance and traffic requirements between nodes into account. The resulting algorithm is able to provide relatively good solutions to the DSNDP in a relatively short period of time.

A small portion of a combined hub location and backbone network design problem is considered in [Shar93]. The authors assume the O-D pair traffic requirements, a set of clusters, inter and intra-cluster topologies, cost and link delay constraints, are given. The authors attempt to: (i) identify the inter-cluster topology and link capacities, and (ii) select gateways in each cluster. The authors aim is to maximise the network reliability to cost ratio. Gateways within each cluster are selected by evaluating all possible combinations of 1 to n (where n is the number of nodes in the cluster) combinations of gateways. The combination that optimises the objective function is chosen as the solution. Although, this exhaustive search based approach will obviously provide good solutions to the problem considered, the problem itself is only a small part of the overall DSNDP.

In [Yan95] the authors describe how a either a greedy ADD or greedy DROP heuristic can be used in the design of ATM networks. The focus of the paper is on how existing network design algorithms can be adapted to design a multi-rate ATM network. Hence, the authors do not provide detailed descriptions of the design algorithms discussed. However, they do outline how the effective bandwidth of multi-rate traffic requirements can be estimated to provide an equivalent single rate traffic requirements matrix as required by traditional backbone network design algorithms. For simplicity we assume that the designer has already used this, or some other, method to determine an input requirements matrix.

In [Cahn96] Cahn considers what he refers to as the Light Network/Server Design Problem (LNSDP). This is very similar to the DSNDP we consider, with two additions. The first is the addition of background peer-to-peer traffic between user nodes. The second is a restriction that the volume of client-server traffic must be a relatively small portion of the total traffic supported by the network. The design procedure in [Cahn96] has two parts. The first part uses the MENTOR algorithm from [Cahn91] and [Kers91] to design a backbone network to support the peer-to-

peer traffic between user nodes. The second part of the procedure uses a greedy DROP heuristic to introduce servers into the network.

An interesting feature of the greedy DROP heuristic employed in [Cahn96] is that when a server is dropped all nodes are reassigned to a single alternate server, effectively merging the two servers. This means that a blind drop algorithm often produces sub-optimal designs. Cahn attempts to overcome this problem by augmenting the DROP heuristic so that before a server is permanently removed, the benefit of removing the server to which its clients will be reassigned is reassessed. It is unclear why Cahn assumes that all clients of a server should be reassigned to a single alternate server. No such restriction appears elsewhere in the literature, and it appears to constrain the design procedure unnecessarily.

As mentioned above, Cahn places a significant restriction on the design procedure, namely that the client-server traffic must be a relatively small proportion of the total network traffic. He then observes that the design procedure will not be suitable for use in networks in which the volume of client-server traffic is comparable to, or dominates, the volume of peer-to-peer traffic. In contrast to the problem posed by Cahn, the DSNDP does not include any peer-to-peer traffic. In a modern network environment it is likely that traffic would be segregated into Virtual Service Networks (VSNs) to ensure quality of service guarantees are honoured. In a VSN dedicated to a DSN style of service there would be little, or no, peer-to-peer traffic, thus it is not included in our examination of the DSNDP. However, as we discuss in Section 6.2.4 the addition of peer-to-peer traffic to the problem would not affect the applicability of our design procedures. Moreover, Cahn's assumption that client-server traffic is a relatively minor portion of total network traffic in a large network is not born out in practice. The last NSFNET traffic measurements of April 1995 showed that client-server traffic (HTTP, FTP, NNTP, SMTP, and DNS), was the source of over 60% of backbone network traffic [NSFNET95].

An additional feature included in the DSNDP that is not included in the client-server portion of the design problem considered by Cahn, is inter-server interaction. Although the inclusion of inter-server traffic complicates the problem it does not necessarily complicate the solution procedure as it simply alters the traffic requirements generated once the number and location of servers is known.

In this section we have seen that while combined hub location and backbone network design problems have been considered in the literature, the complete solution of a problem such as the DSNDP has not been attempted. Despite this, as was discussed at the beginning of this section, the similarities between the problems considered in this section and the DSNDP are sufficient to suggest that a number of the solution procedures described here may be applicable to the design of DSN's. Hence we develop a node clustering heuristic to solve the DSNDP in Section 5.2. We also develop a combined ADD and DROP heuristic in Section 5.3.1, and a modified ADD-*k* backbone node selection heuristic from [Gavi92a] in Section 5.3.2.

3.3.3 Distributed Database and Network Topology Design

We have already observed that DSN's are required to support distributed database style applications. A number of procedures to solve similar distributed database design problems have appeared in the literature. A general form of the distributed database design problems appearing in the literature can be described as follows: given a number of user nodes, a set of databases that the users wish to query, and/or update, the rates at which they do so, the storage requirements of each database, the processing power required to perform query and update operations, the cost of various capacities (processing and storage) of servers, and network communication cost: Determine the number and location of servers, the number of copies of each database, and the assignment of databases (and hence associated users) to servers that will support the required service(s) at minimum cost. The problem may also require that the topology and capacity of the communication network employed to connect users to database servers, and database servers to one another be determined. The design may be subject to link and/or server capacity and delay constraints. User queries are usually assumed to be delivered to the single copy of the database they are assigned to, while updates are delivered to all copies of the database (either directly, or indirectly via the database server that the user is assigned to).

As discussed in Chapter 1, the DSN can be considered as a distributed database system in which there are multiple copies of a single database, or a single database partitioned amongst the servers. It is assumed that in a DSN all user query and update traffic is delivered to the server to which they are assigned. Although in our formu-

lation of the DSNDP we consider the design of a network to support only one service (i.e. one distributed database), as we discuss in Section 7.5, our solution procedures for the DSNDP can easily be extended to include multiple services.

From our review of network design problems in previous sections it can be seen that, like the DSNDP, the general distributed database problem involves aspects of location-allocation and network design problems. The complexity of this type of problem has meant that most previous work has focused on either the design of network topologies, or the location-allocation of databases separately. The design of network topologies has been discussed in Sections 2.5 and 2.6. A number of researchers have considered distributed database location-allocation problems that do not include the design of the supporting network, see for example [Levi78], [Lee95], [Gavi85b], [Pirk86], [Gavi86a], [Gavi87], [Gavi92c] (and the references therein). Others, such as [Chu69], have considered the effect of fixed network topologies on the distribution of databases. In [Chen80] link capacities are included in the design, but nodes are restricted to being directly connected to one another. To summarise, although the design of distributed databases systems has been approached in the literature, most researchers have concentrated on the distribution of databases and have largely ignored the design of the supporting network.

The DSNDP includes aspects from a variety of network design problems, including: hub and facility location, centralised and backbone network design. Despite the previous work on similar problems, for example combined hub location and backbone network design and distributed database design, the complete design of a DSN has not been addressed in the literature. However, a number of solution approaches employed in previous work can be adapted to aid in the solution of the complete DSNDP, as discussed in more detail in the following section.

3.4 Problem Decomposition & Solution Approaches

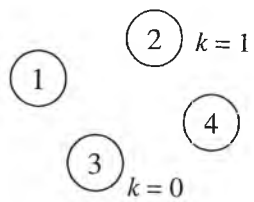
A characteristic feature of DSNDP's is that a two dimensional O-D pair traffic requirements matrix is not available to the designer. This means that although the volume of traffic produced by each client node is known, its destination is not. Traffic destinations are known only when the number and location of servers, along with

the assignment of client nodes to servers are known, that is when a two dimensional O-D pair traffic requirements matrix can be generated. Once this information is available a suitable TCFA design algorithm can be applied to design the network topology connecting clients to servers and servers to one another.

The search for the optimal design of a DSN can be considered as a search for the configuration of clients and servers that minimises the cost of the resulting network. For a network of n nodes (all of which are potential server locations) there are

$$\sum_{k=1}^{n} \binom{n}{k} = 2^n - 1 \qquad (3.22)$$

possible server configurations. If we assume that client nodes are assigned to their nearest server then the number of possible solutions is $O(2^n)$ (see Equation (3.22)). However, if client nodes are allowed to be assigned to any server in the network there are k^n possible assignments of clients to servers for each configuration of k servers. This can be seen by observing that the assignment of client nodes to servers can be represented as an n digit base k number, as illustrated in Figure 3.2.



A network of 4 nodes ($n = 4$) with 2 potential server locations ($k = 2$).

$n = 1$	2	3	4
0	1	0	1

Nodes 1 and 3 assigned to server 0, nodes 2 and 4 to server 1.

$n = 1$	2	3	4
1	1	0	0

Nodes 1 and 2 assigned to server 1, nodes 3 and 4 to server 0.

$n = 1$	2	3	4
0	0	0	0

All nodes assigned to server 0.

Figure 3.2 Representation of client node assignments as n digit base k numbers.

This yields an $O(n^n)$ solution space, or

$$\sum_{k=1}^{n} \frac{n!k^n}{k!(n-k)!} \tag{3.23}$$

possible client-server configurations if clients are allowed to be assigned to any server in the network, and all nodes are potential server locations. In either case the search for the optimal DSN design is a *combinatorial optimisation problem* with the evaluation of each potential solution involving the design of the supporting network topology. As we shall see in Chapter 4, the design of the network topology (the Concave TCFA problem) is itself a concave, non-linear minimisation problem.

3.4.1 Global Search Techniques

As we have observed, the DSNDP can be viewed as either a concave MINLP minimisation problem, or a combinatorial optimisation problem involving the repeated solution of a Concave TCFA problem. The concave nature of the problem leads us to examine global optimisation techniques to solve it.

The most obvious approach to finding the optimal solution is to perform an exhaustive search of the entire solution space. Unfortunately, even if the solution space is limited to configurations of clients and servers, an exhaustive search of the solution space is impractical for networks with more than approximately 10 nodes. This is because the number of possible solutions is either $O(2^n)$ or $O(n^n)$ with the number of nodes, n, depending on whether we assume client nodes are assigned to their nearest server, or not. In either case the size of the space grows rapidly with the size of the network. In addition, the evaluation of each potential solution involves the solution of a Concave TCFA problem to design the network topology. Hence the evaluation of each possible solution is an expensive process.

A number of global optimisation procedures have appeared in the literature that do not involve an exhaustive search of the solution space. These procedures can be classified as either stochastic or deterministic.

Stochastic methods, such as Simulated Annealing (SA) [Kirk83] [Davi87] and Genetic Algorithms (GA) [Davi87] [Beas93b], do not make any assumptions about the solution space. They incorporate random elements into their search procedure

and converge to the global optimum with a probability approaching one as their running time approaches infinity [Davi87].

Both the SA and GA techniques rely on the researcher defining a set of functions (called "neighbourhood" functions in SA and "mutation" functions in GA) that allow the procedures to generate new "random" solutions more, or less, related to the current solution depending on the current state of the system. Both techniques then rely on generating and evaluating a large number of potential solutions to narrow their searches to the most promising region of the solution space.

Unfortunately, these features make both of these optimisation procedures inappropriate for the problem at hand. If we allowed a SA or GA procedure to search the solution space by altering link and server capacities the solution space would be enormous and the procedure would have to solve a constrained, multicommodity, minimum cost flow problem for each generated solution. If we restricted the procedures to selecting client and server configurations then a Concave TCFA problem has to be solved to evaluate each generated solution. In either case the evaluation of each potential solution would be expensive, thus limiting the number of solutions that could be examined and hence the effectiveness of both the SA and GA approaches.

3.4.2 Local or Directed Search Techniques

Given that global search techniques (exhaustive or random) are impractical for DSN design, we consider the use of local search procedures. In Chapter 5 we examine three local search procedures for solving the DSNDP. The first of these procedures is a node clustering algorithm designed using assumptions about the expected form of the optimal solution. The remaining two procedures are based on "greedy" searches of the server configuration solution space which do not rely on any assumptions about the form of the optimal solution. As we saw in Chapter 2, these types of search procedures are commonly employed to solve hub and facility location problems.

Two out of three of the procedures are able to solve reasonably large design problems (up to 50 potential server locations) because they examine a relatively small

portion of the solution space. However, the best deterministic search technique to use is an open question which we address in Chapter 5.

All three procedures search the server configuration solution space. This means that they must all solve a Concave TCFA network design problem to evaluate each potential solution examined. As seen in Chapter 2 most existing work in the areas of facility or hub location does not deal with the design of the network topology required to connect client nodes to servers, and servers to one another. Rather it is assumed that a set of connection, or communication, cost coefficients can be supplied for each O-D node pair in the network. In reality network communication costs depend on aspects such as the topology of the network links, the volume of traffic on each link, the additional capacity and/or links required to satisfy quality of service and reliability constraints and so on. Given a set of nodes and an O-D node pair traffic requirements matrix the TCFA problem is to determine the topology, capacity of network links and routing of traffic on them to satisfy the traffic requirements and any additional constraints at minimum cost.

As seen in Chapter 2, the solution of TCFA problems with strongly concave link cost functions (as we have assumed) is not adequately addressed in the literature. In Chapter 4 we develop a Concave Link Elimination (CLE) algorithm to efficiently solve Concave TCFA problems.

3.5 Conclusions

In this chapter we have formulated the DSNDP as a concave MINLP. This formulation illustrates how the DSNDP combines aspects of the network design problems examined in Chapter 2. This led us to examine previous work on similar network design problems that also combine multiple problems from Chapter 2. Despite the similarities between the problems considered in previous work, to our knowledge the complete design of a DSN has not been adequately addressed by previous work.

We have observed that the design problem can be viewed as a combinatorial optimisation problem in which each configuration of servers and assignment of client nodes to servers is a potential solution. We have shown that the solution space of the combinatorial optimisation problem is either $O(2^n)$ or $O(n^n)$ with the number of

nodes, n, depending on whether or not we assume client nodes are assigned to their nearest server. In addition, the evaluation of each potential solution involves the solution of a Concave TCFA problem to design the network topology.

We have shown that global search techniques, such as SA and GA, are not suited to solving the DSNDP, due to the large number of potential solutions that they must examine and the expense associated with evaluating each solution.

We have also discussed the applicability of local or directed search techniques, which do not examine large portions of the solution space. Chapter 5 examines three procedures, which all assume that client nodes are assigned to their nearest server and operate by searching the server configuration solution space. Finding the optimal solution relies on the server configuration space being relatively well behaved, and requires the solution of a Concave TCFA network design problem to evaluate each potential solution. We examine procedures for solving Concave TCFA problems in the next chapter.

3.6 Deficiencies & Developments

Having completed our review of previous work and possible approaches to addressing the DSNDP, we summarise the deficiencies in previous work relating to the DSNDP, and the developments presented in this thesis.

3.6.1 Deficiencies in Previous Work

1. Neither hub nor facility location problems include the design of the network topology connecting customer or client nodes to hubs or facilities. It is assumed that non-hub (or facility) nodes will be connected directly to hubs, and that hubs will be connected directly to one another. A matrix specifying the fixed, or variable cost of connecting any pair of nodes is assumed given. In reality the cost of connecting any two nodes is likely to be strongly dependent on the topology of links and interaction of node pairs, and hence the volumes of traffic on links. This dependence cannot be captured in an O-D pair communication cost matrix specified apriori.

2. Similarly, the previous work on hub and facility location problems has

assumed that the cost of enabling a pair of nodes to interact directly with one another is either, fixed (regardless of the level of interaction), or an offset-linear function of the level of interaction. Again, communication costs often exhibit economies of scale with respect to the level of interaction, or traffic volumes. These economies of scale in communication (i.e. link) costs cannot be captured by fixed, or offset-linear, cost functions.

3. The DSNDP is similar to the facility location and access network design problems in that the client node traffic requirements are specified rather than an O-D pair requirements traffic matrix. However, previous facility location and access network design problems do not allow for interaction between facilities (i.e. servers).

4. Backbone network topology design algorithms assume that an O-D pair traffic requirements matrix is supplied. The facility (i.e. server) location aspect of the DSNDP means that the O-D pair traffic requirements are not available apriori. This means that network topology design algorithms cannot be employed directly.

5. Existing network topology design algorithms are not well suited to solving problems in which the link cost function is strongly concave and there are a large number of possible link capacities available.

6. Combined hub location and backbone network design problems assume simple access network topologies (i.e. star). As the cost of the access networks may be significantly affected by their topology, the existing combined network design algorithms may produce significantly sub-optimal designs.

7. The majority of previous work on network design example problems has been generated by assuming uniform traffic requirements between all node pairs. This assumption produces well balanced network designs which may not be appropriate in practice. This also means that existing algorithms may have been (inadvertently) biased towards balanced network solutions.

8. Previous work on distributed database problems does not adequately address the design of a DSN. This is because in most cases the design of the supporting network topology is ignored.

In summary, DSNDP combines aspects of a number of problems that have been addressed in the literature. However, previous work does not completely address the complete design of a DSN.

3.6.2 Developments in this Dissertation

This section describes the developments presented in this dissertation that enable the design of DSN's. We also note how the deficiencies in previous work, discussed above, are addressed.

1. The DSNDP is formulated as a mathematical programming problem which combines aspects of both facility location and network design problems. Servers (i.e. facilities) must be introduced to satisfy the given client-server traffic requirements vector, and in addition, the supporting topology network must be designed. A key observation is that the problem consists of two types of network traffic, client-server and inter-server, whose interaction is determined by the location of servers and the assignment of clients to servers. This observation leads to a solution approach that treats the problem as a two stage location-allocation and network design problem. That is, solutions may be generated by first determining the location of servers and allocation of clients to them, and then designing the supporting network topology. Once the location of servers and the allocation of clients is known, the client-server traffic requirements vector can be translated into an O-D pair traffic matrix, and the supporting network topology designed. This means that the cost of connecting nodes together is dependant on both the traffic volume transferred between them and the route it follows. See points 1, 3, 4 and 6 above.

2. A feature of the mathematical programming formulation of the DSNDP is the use of a "double power-law" cost function for both links and servers, rather than a fixed or offset-linear cost function. The double power-law cost function is an extension of the "power-law" cost function in [Klei76] and [Gerl77]. Concave cost functions of this form have been used extensively in network design to capture the effects of economies of scale in the pricing of network components. The advantage of the double power-law function is that it is able to approximate a wide variety of link cost values. Moreover, the

double power-law function is "smooth." The constant offset in Kleinrock and Gerla's power-law cost function, [Klei76][Gerl77], means the solution surface includes step changes as links are added or removed. This means that with our double power-law cost function more powerful gradient based local search techniques, such as Sequential Quadratic Programming (SQP), that rely on a smooth solution surface can be employed in the search for optimal solutions (see Sections 4.4 and 5.5.2). In addition, the double power-law cost function reduces the number of binary variables in the problem formulation. This is because the double power-law cost function does not include a fixed offset component. This eliminates the need for a binary variable associated with each link (or server) to include the initial cost of each active link in the objective function. The reduction in the number of binary variables simplifies the problem formulation and solution considerably.

3. In Chapter 4 a Concave Link Elimination (CLE) procedure is developed to solve network topology design problems which involve strongly concave link cost functions, referred to as the Concave TCFA problem. As discussed in Chapter 2, existing algorithms for the TCFA problem do not cope well in the context of strongly concave link cost functions and a large number of possible link capacities (see point 5 above). The only procedure in the literature that are applicable to concave link cost functions are Kleinrock and Gerla's *Concave Branch Elimination (CBE)* procedure, [Klei76] [Gerl77], Gavish *et al.*'s MILP based procedure, and a greedy link elimination procedure due to Gersht and Weihmayer [Gers90]. Our results (see Chapter 6) show that the CLE procedure is able to produce network designs whose cost is within 1% (on average) of those produced by Gersht's procedure [Gers90], in significantly less time (the CLE procedure is almost two orders of magnitude faster than Gersht's procedure in designing 20 node networks). When compared to the CBE procedure, [Gerl77][Klei76], the cost of the designs produced by the CLE procedure are 45% (on average) cheaper, and again take less time to produce.

4. In addition to developing the CLE procedure, a lower bounding procedure is also presented in Chapter 4. The lower bounding procedure is based on a continuous branch-and-bound algorithm from [Ryoo95].

5. In Chapter 5 three local search procedures are developed to search for the optimal number and location of servers, and assignment of clients to them. One procedure is based on a node clustering algorithm, while the other two are based on greedy search heuristics. All three procedures employ the CLE procedure developed in Chapter 4 to design the network topology required to evaluate each configuration of servers and clients (see points 6 and 8 above).

6. As in Chapter 4, a lower bounding procedure for the DSNDP is presented in Chapter 5. This lower bounding procedure also uses the continuous branch-and-bound algorithm from [Ryoo95]. However, the lower bounding problem is developed by simplifying the full DSNDP.

7. As noted in Chapter 2, previous network design algorithms have been tested using either only a few networks and/or uniform, balanced O-D pair traffic requirements. Either of these features can lead to design algorithms being (inadvertently) biased towards producing particular types of solutions. To avoid this problem we present an extensive performance comparison of our network design algorithms using a large number of randomly generated network examples are presented in Chapter 6. To generate realistic O-D pair traffic requirements for each network we have assumed that the level of interaction between nodes is proportional to the user population located at each node and inversely proportional to the distance between them. A procedure for generating traffic requirements based on this assumption is described in Chapter 6.

8. Chapter 6 also presents a design methodology that describes how the full DSNDP can be solved using the procedures described in the previous chapters.

4. The Concave TCFA Problem

Greed is good. . . greed is right. . . greed works. . . greed clarifies, cuts through, captures the essence. . .

- Gordon Gekko, Wall Street.

4.1 Introduction

In this chapter we examine the *Concave Topology Capacity and Flow Assignment (TCFA)* problem. The bulk of this chapter also appears in [Stac97b]. As discussed in Section 2.6, only three algorithms in the literature are appropriate for solving TCFA problems with concave link cost functions, Kleinrock and Gerla's *Concave Branch Elimination (CBE)* procedure, [Klei76] [Gerl77], Gavish *et al.*'s MILP based procedure, and a greedy link elimination procedure due to Gersht and Weihmayer [Gers90]. Again as discussed in detail in Section 2.6, none works well in practice. Our experiments (see Section 4.7.2, [Stac97a], and [Stac97b]) show CBE procedure does not perform well in the context of strongly concave link cost functions. Gavish *et al.*'s MILP based procedure relies on there being only a few possible link capacities to select from, an assumption that is untrue in an ATM based environment. While Gersht's algorithm performs well, it is too slow for use on anything but small network design problems.

In this chapter we present a *Concave Link Elimination (CLE)* procedure, based on Gersht's greedy link elimination procedure. In Section 4.7.2 our algorithm is shown to perform at least as well as Gersht's procedure and to be faster than both the CBE and Gersht procedures. Gavish *et al.*'s MILP based procedure is not considered because of its inability to cope with a large number of possible link capacities. In addition, we formulate a lower bounding problem which we solve using a continuous lower bounding procedure to assess the quality of the design procedures.

In Section 3.4 we observed that the design of a DSN can be viewed as a search of the possible configurations of servers. A configuration of servers specifies the number and location of servers and the assignment of clients to them. The evaluation of each configuration of servers requires the design of the network connecting clients to servers and servers to one another. The design of such a network involves determining which links to include (the topology), their capacity and the routing of traffic in the network. The objective is to find a minimum cost solution which is able to support the required origin-destination (O-D) traffic demands while satisfying all constraints. Kleinrock and Gerla [Klei76] [Gerl77] refer to this problem as the Topology, Capacity and Flow Assignment (TCFA) problem.

As discussed in Section 3.2.4 we assume that link cost functions are concave to capture the effect of economies of scale in link capacity pricing. We refer to a TCFA problem that involves concave link cost functions as a Concave TCFA problem and examine algorithms to solve this problem in this chapter.

A number of methods to solve the TCFA problem (in various forms) have appeared in the literature, the key approaches are reviewed in Section 2.6. Our survey of the TCFA literature led us to conclude that only Kleinrock and Gerla's CBE procedure [Klei76] [Gerl77], and Gersht's link elimination procedure [Gers90], are designed to cope with general concave link cost functions. Although Gavish et al. [Gavi86b] [Gavi89] [Gavi90] [Gavi92a], and Dutta's [Dutt92] solution procedures cope with concave link cost functions they are limited to environments in which the number of possible link capacities is small. In order to design a network topology given a set of server locations and the assignment of user nodes to them (i.e. an O-D traffic requirements matrix) we have developed our own Concave Link Elimination (CLE) procedure, based on Gersht's procedure.

Before examining either the CLE or CBE procedures in detail we formulate the Concave TCFA problem as a MINLP in Section 4.2. We then reformulate the problem, in Section 4.3, to form a less complex NLP whose solution cost is a lower bound on the cost of the optimal solution of the full problem, as described in Section 4.2. In Section 4.4 we describe a continuous branch-and-bound algorithm to solve the lower bounding problem.

In Section 4.5 we briefly examine Kleinrock and Gerla's *Concave Branch Elimination (CBE)* procedure [Klei76] [Gerl77]. Although the CBE procedure is designed to handle concave link cost functions the function parameters used by Kleinrock and Gerla in [Klei76] and [Gerl77] produce almost linear link cost functions. Our experiments have shown that the network designs produced by the CBE procedure when used with strongly concave link cost functions would often include an unnecessarily large number of links and could easily be improved upon by hand. This led us to investigate Gersht's greedy link elimination procedure [Gers90]. While Gersht's procedure performs well in the presence of concave link cost functions (indeed it relies on them) it is too slow for use on anything but small network design problems. In Section 4.6 we describe a Concave Link Elimination (CLE) procedure, based on Gersht's greedy link elimination procedure, which performs at least as well and considerably faster than either the CBE or Gersht procedure.

The three heuristic design procedures (CBE, Gersht and CLE) are compared against the bounds produced by the continuous branch-and-bound algorithm in Section 4.7.

4.2 Mathematical Programming Formulation

In this section we formulate the TCFA problem as a multi-commodity flow problem. The network to be designed is modelled as a directed network $G = (N, L)$, where N is the set of network nodes and L the set of links. Capacity, c_{ij}, is assigned to each link $(i, j) \in L$ to allow it to carry traffic in either direction (i.e. $c_{ij} = c_{ji}$). A link with zero capacity is effectively removed from the network. The assignment of capacity to a link incurs both a "termination" cost proportional to the capacity and a "line" cost proportional to both the capacity and length of the link, see Equations (3.4) and (3.5). As discussed in (3.2.4), we use a *double power-law* link cost function to model the economies of scale that exist in the pricing of communication network links.

Table 4.1 shows the costs assumed in our experiments for a range of link capacities. They are based on figures supplied by a major telecommunication provider operat-

ing in Australia[1]. For the purposes of assigning link capacities in the network design we assume that link capacity is available in 2 Mb/s increments. Table 4.1 shows the resulting parameter values when the double power-law cost function (Equations (4.1) and (4.2)) is fitted to the component cost data from Table 4.1. .

Table 4.1 Link Capacities and Corresponding Costs

Capacity (Mbps)	Termination Cost ($/year)	Line Cost ($/year/km)
2	1,750	40
10	2,800	50
34	4,800	55
155	10,000	80
300	14,000	90
622	21,000	120

Table 4.2 Double power-law function (Equations (4.1) and (4.2)) parameters fitted to cost data.

	β_i	α_i	β_{i+1}	α_{i+1}
Link Termination Cost (i = 1)	1000	0.15	400	0.595
Link Line Cost (i = 3)	37.37	0.09149	0.3606	0.7722

Typical additional constraints on the network design include restrictions on the allowable link capacities, performance and reliability constraints. We assume that link capacities must be selected from a set of predetermined values, Q, and that maximum capacity limits, C_{ij}, exist for each link. Typical performance related constraints include a limit on the maximum average queuing delay, T_{max}, in the network [Klei76] [Gerl77] [Dutt92], and a limit on the number, or total length, of links in the path between origin and destination nodes. Protecting the network against link failures can be achieved by constraining the minimum degree of each node producing a k-connected network. For simplicity we do not include constraints on the path length or connectivity of the network in our mathematical formulation of the problem. The ability to enforce these types of constraints is often included in design algorithms by testing for the violation of path length, or connectivity constraints when the impact of the removal of a link from the network is evaluated.

1. The figures were supplied under the terms of a Non-Disclosure Agreement and we are unable to name the company concerned.

Viewing the TCFA problem as a multi-commodity flow problem, each node in the network can be considered as the source of a commodity, varying volumes of which must be delivered to the other nodes in the network. Each commodity, $k \in K$, has a single source node s^k, a set of destination nodes $d_i^k, (i \in N - s^k)$ and a volume of traffic, γ_i^k, to be delivered from s^k to each destination d_i^k. Let $\gamma^k = \sum_{i \in N} \gamma_i^k$ be the total traffic for commodity k, and $\gamma = \sum_{k \in K} \gamma^k$, the total traffic requirement of the network. Also, let x_{ij} denote the total traffic flow on link (i, j), and x_{ij}^k the total flow of commodity k on link (i, j). Using this notation the TCFA problem is formulated as follows:

$$\min\ Z(c) = \sum_{(i,j) \in L} \left\{ \beta_1 c_{ij}^{\alpha_1} + \beta_2 c_{ij}^{\alpha_2} + \left(\beta_3 c_{ij}^{\alpha_3} + \beta_4 c_{ij}^{\alpha_4} \right) l_{ij} \right\} \tag{4.1}$$

Subject to:

$$\sum_{\{j \ni (i,j) \in L\}} x_{ij}^k - \sum_{\{j \ni (j,i) \in L\}} x_{ji}^k = \left\{ \begin{array}{ll} \gamma^k & \text{if } i = s^k \\ -\gamma_i^k & \text{if } i = d_i^k \\ 0 & \text{otherwise} \end{array} \right\} \forall i \in N, k \in K \tag{4.2}$$

$$\lceil x_{ij} + x_{ji} \rceil_Q \leq c_{ij}, \forall (i,j) \in L \tag{4.3}$$

$$\frac{1}{\gamma} \sum_{(i,j) \in L} \left(\frac{x_{ij} + x_{ji}}{c_{ij} - (x_{ij} + x_{ji})} \right) \leq T_{max}, \forall (i,j) \in L \tag{4.4}$$

$$c_{ij} \leq C_{ij}, \forall (i,j) \in L \tag{4.5}$$

$$x_{ij} \geq 0, c_{ij} \geq 0, \forall (i,j) \in L \tag{4.6}$$

where $\lceil y \rceil_Q$ is a ceiling operator over the set of allowable link capacity values, Q. That is, $\lceil y \rceil_Q$ denotes the smallest value from Q, not less than y.

Constraint (4.2) is the standard node conservation of flow for each commodity k. Constraint (4.3) ensures that the capacity assigned to a link is greater than the traffic on the link, while (4.5) ensures the maximum allowable capacity for each link is not

exceeded. Constraint (4.4) employ's Kleinrock's expression for the maximum average queuing delay ([Klei76], Equation. 5.19), and ensures that the capacity assigned to each link is such that the maximum average queuing delay does not exceed the maximum, T_{max} (as specified by the network designer). Constraint (4.6) ensures non-negativity in the decision variables.

As formulated, the TCFA problem is a non-linear, combinatorial optimisation problem. Non-linearities are introduced into the problem by both the objective function (4.1) and the performance constraint (4.4). In addition, the ceiling function in the link capacity constraint introduces a discrete component, creating a combinatorial optimisation problem.

4.3 Reformulating to Find a Lower Bound

The Concave TCFA problem is complicated by the non-linear constraints, (4.3) and (4.4). Our first step toward formulating a lower bound on the cost of the full problem is to linearise the constraints. Ideally we would like to remove all non-linear constraints and decompose the multi-commodity problem into a number of single commodity problem each of which is relatively easy to solve.

Constraint (4.3) on the allowable link capacity, is made non-linear by the ceiling function, forcing link capacities to be selected from a discrete set Q. This constraint can be linearised by simply removing the ceiling function, thus allowing any link capacity (less than the maximum) to be selected.

Constraint (4.4), on the maximum allowable link queuing delay, is non-linear in the capacity and flow variables. However, Dutta [Dutt92] provides a way that this constraint can be reformulated such that it is linear. Dutta observes that most network design algorithms (including [Klei76] [Fran72] [Gerl77] [Ng87]) attempt to achieve uniform utilisation of links, that is, they strive to avoid under or over utilised links. Dutta reformulates the constraint on the average packet delay as a constraint on link

utilisation by distributing the delay requirement equally over all active links. Thus Constraint (4.4) becomes:

$$\frac{x_{ij}+x_{ji}}{c_{ij}-(x_{ij}+x_{ji})} \leq \frac{\gamma T_{max}}{|L'|}, \forall (i,j) \in L \tag{4.7}$$

or equivalently,

$$\Psi(x_{ij}+x_{ji}) \leq c_{ij}, \forall (i,j) \in L \tag{4.8}$$

where Ψ is a constant link utilisation factor and L' is the set of all active links.

$$\Psi = \frac{|L'|}{\gamma T_{max}} + 1 \tag{4.9}$$

The notation $|L'|$ denotes the size, or number of elements, in the set L'.

Dutta's results show that using a uniform link utilisation constraint to assigned link capacity causes relatively little excess capacity to be installed. In [Dutt92] the actual average network delay was found to be no more than 10% below the stipulated limit. In our experiments we refined Dutta's reformulation by using the set of only those links that are active rather than the set of all links, L, in constraints (4.7) and (4.9). Since non-active links do not contribute to the delay in the network, altering constraint (4.9) to exclude them allows the link utilisation factor, Ψ, to more accurately approximate the desired delay limit. Our experiments have shown even closer correspondence between the target and achieved delay limits than that reported by Dutta in [Dutt92]. Unfortunately, using the set of active, rather than all, links makes constraint (4.8) non-linear, so for the purposes of this exercise we use Dutta's original reformulation method.

While Dutta's reformulation of the link delay constraint introduces very little excess capacity into final network designs in practice, it does mean that (4.7) is a slightly tighter constraint on the solution space than (4.4). This means that there is a chance, albeit small, that the cost of the optimal solution to this lower bounding problem may be slightly greater than the cost of the optimal solution to the unrelaxed problem. As our results below show, the gap between the cost of the best solution we are able to obtain to the relaxed problem and the cost of the best feasible solutions we can generate rapidly becomes larger than any over-estimation introduced by the

reformulation as the size of the network increases. Thus we do not feel that the use of Dutta's reformulation of constraint (4.4) detracts significantly from the usefulness of the lower bound described above.

Using this reformulation of the link delay constraint, both (4.3) and (4.4) can be replaced by a single constraint on link utilisation:

$$\Psi \sum_{k \in K} (x_{ij}^k + x_{ji}^k) \leq c_{ij}, \forall (i,j) \in L \tag{4.10}$$

This new link utilisation constraint (4.10) is now the only "bundling" constraint tying the commodity flows together. The most common approach to developing lower bounds on multi-commodity flow problems is to use Lagrangian relaxation to "unbundle" the commodity flows. By introducing additional variables (Lagrangian multipliers) and relaxing the bundling constraints into the objective function, the problem can often be made "separable". That is the problem can be treated as the sum of a number of single commodity flow problems, each of which is easier to solve than the multi-commodity flow problem as a whole. Following this approach we associate non-negative Lagrangian multipliers, w_{ij}, with (4.10) thus bringing it into the objective function, creating the following Lagrangian sub-problem ($\inf(c, x)$ denotes the minimum of the following expression with respect to c and x):

$$\begin{aligned} L(w) = \inf(c,x) & \sum_{(i,j) \in L} \left\{ \beta_1 c_{ij}^{\alpha_1} + \beta_2 c_{ij}^{\alpha_2} + \left(\beta_3 c_{ij}^{\alpha_3} + \beta_4 c_{ij}^{\alpha_4} \right) l_{ij} \right\} \\ & + \sum_{(i,j) \in L} w_{ij} \left(\Psi \sum_{k \in K} (x_{ij}^k + x_{ji}^k) - c_{ij} \right) \end{aligned} \tag{4.11}$$

or, equivalently:

$$\begin{aligned} L(w) = \inf(c,x) & \sum_{(i,j) \in L} \left\{ \beta_1 c_{ij}^{\alpha_1} + \beta_2 c_{ij}^{\alpha_2} + \left(\beta_3 c_{ij}^{\alpha_3} + \beta_4 c_{ij}^{\alpha_4} \right) l_{ij} \right\} \\ & + \sum_{k \in K} \sum_{(i,j) \in L} w_{ij} \Psi (x_{ij}^k + x_{ji}^k) - w_{ij} c_{ij} \end{aligned} \tag{4.12}$$

Subject to:

$$\sum_{\{j \ni (i,j) \in L\}} x_{ij}^k - \sum_{\{j \ni (j,i) \in L\}} x_{ji}^k = \left\{ \begin{array}{ll} \gamma^k & \text{if } i = s^k \\ -\gamma_i^k & \text{if } i = d_i^k \\ 0 & \text{otherwise} \end{array} \right\} \forall i \in N, k \in K \quad (4.13)$$

$$c_{ij} \leq C_{ij}, \forall (i,j) \in L \quad (4.14)$$

$$x_{ij} \geq 0, w_{ij} \geq 0, \forall (i,j) \in L \quad (4.15)$$

Unfortunately, (4.12) remains inseparable over the commodities due to the power-law nature of the link cost function. This means that, a Lagrangian relaxation of the bundling constraint does not allow us to decompose the Concave TCFA multi-commodity flow problem into a collection of single commodity flow problems. Thus, we are unable to use Lagrangian relaxation to further simplify the lower bounding problem.

Despite relaxing a number of the non-linear components of the Concave TCFA multi-commodity flow problem it cannot be decomposed into a collection of single commodity flow problems using Lagrangian relaxation. Thus, we use a continuous branch-and-bound algorithm, to solve the simplified Concave TCFA problem, (4.1) subject to (4.2), (4.5), (4.6) and (4.10) directly.

4.4 Continuous Branch-and-Bound

Our goal is to determine a guaranteed lower bound on the cost of the optimal solution to Concave TCFA problems. Having formulated a simplified, non-convex, minimisation problem whose optimal solution cost is known to be a lower bound on the cost of the optimal solution to the primal problem, we have two options. Ideally, we would like to solve the lower bounding problem to optimality and so determine a lower bound to the primal problem. If, however, solving the lower bounding problem to optimality is not possible (non-convex minimisation problems are often very difficult to solve to optimality), we need to determine a lower bound on the cost of the optimal solution to the lower bounding problem.

As mentioned in Chapter 2, non-convex minimisation problems are often tackled using stochastic techniques such as Simulated Annealing (SA) [Kirk83] [Davi87] or Genetic Algorithms (GA) [Beas93a] [Davi87]. Given infinite time these methods are guaranteed to find the optimal solution to the problem. In finite time, however, they are able to find only feasible solutions to the problem without any guarantee of how far they are from the optimal solution [Davi87]. While in practice methods such as SA and GA have been shown to perform very well, we are interested in determining a guaranteed lower bound on the cost of the optimal solution to a given problem. Hence, these techniques are not applicable for determining a guaranteed lower bound on the cost of the optimal solution to a problem.

In contrast, deterministic methods take advantage of the mathematical structure of the problem and often guarantee finite convergence within a specified level of accuracy [Hors93]. Deterministic methods include branch-and-bound, cutting plane algorithms and decomposition schemes, with branch-and-bound being the most widely used. These methods are described in detail in [Nemh88] and [Hors93].

To find a lower bound on the cost of an optimal solution to the Concave TCFA problem we have chosen to use a deterministic global optimisation algorithm, based on a continuous branch-and-bound search of the solution space, presented in [Ryoo95]. The major advantage of the continuous branch-and-bound approach is that guaranteed lower and upper bounds are available at any time during the search for the global optimum. This means that the branch-and-bound algorithm can be used to both solve small problems to optimality, and to provide upper and lower bounds on the cost of the optimal solution to larger problems, which can not be solved to optimality due to their complexity.

The continuous branch-and-bound algorithm we employ is explained in detail in [Ryoo95], we describe it briefly here.

The algorithm relies on the ability to formulate a relaxation R of the non-convex minimisation problem P by either enlarging the feasible region and/or underestimating the objective function of P. The relaxation is constructed in such a way that the difference between the optimal objective function values of P and R is a non-increasing function of the size of the feasible region over which the relaxation is developed. That is, as the region of interest is restricted, the difference between P

and R decreases. Moreover, the relaxations used can typically be solved to their respective global minima using conventional minimisation techniques (that is, they are typically convex minimisation problems) [Ryoo95].

Observe that the objective function of the Concave TCFA problem (4.1) is the sum of concave functions of single variables (i.e. it is separable). This means that a convex underestimating function R of P can be generated by formulating the sum of linear underestimating functions of each component of (4.1). Before discussing the formulation of the underestimating functions we illustrate the operation of the method in minimising a non-convex function in one variable, x, with the aid of Figure 4.1 (adapted from [Ryoo95]).

4.4.1 Continuous Branch-and-Bound in Operation

In Figure 4.1(a) the curve P represents a non-convex function of a single variable. The curve labelled R represents a convex relaxation of P over the domain of interest (which in this case spans the entire x-axis). The minimum value of R can easily be found using any convenient minimisation technique and is marked as L in the figure ($L = R(x)$). The value L represents a guaranteed lower bound on the minimum value of P. Using the relaxed solution as a starting point, local minimisation techniques can be used to obtain a valid upper bound, U, for P (see Figure 4.1(b), in which a local, gradient based search is started from $P(x)$ to find a better upper bound, U). The global minimum is known to lie between U and L. If U and L are sufficiently close to one another the algorithm terminates. If not, the domain of x is subdivided in two about x. This operation results in the creation of two new minimisation problems whose domains are guaranteed to be smaller than their parent's (see Figure 4.1(c)). New underestimating functions are formulated for each of the two new subdivisions and the relaxed problems solved to provide a lower bound on the primal function within each sub-domain. This process of subdividing the domain is referred to as "branching". The process is then repeated for each subdivision until the subdivisions possess lower bounds that either exceed or are sufficiently close to the best found feasible solution of problem P. This leads to a method for searching over a tree whose nodes correspond to subdivisions of the solution space, see Figure 4.1(d). At any time during the search, parts of the solution space (tree nodes) can be excluded from further consideration by comparing their respective lower

bounds to the current upper bound (e.g. node $R2$ in Figure 4.1(d)). At all times the global minimum is bounded between the lowest lower bound L and the value U of the best feasible solution found.

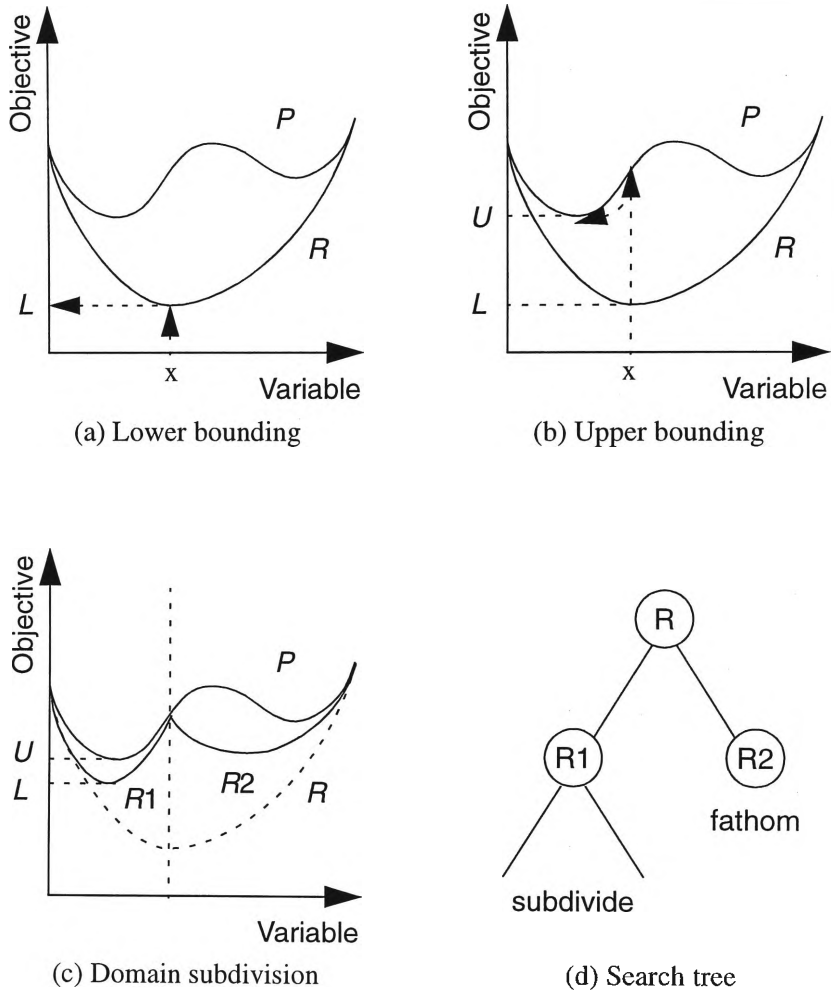


Figure 4.1 Continuous branch-and-bound in operation.

The minimisation of functions of more than one variable complicates the branching procedure and adds to the size of the search tree. In the presence of multiple variables each subdivision (node) is still divided in two in the branching phase. However, the branching is based on the value of the variable that contributes most to the difference between the relaxed and primal objective functions, R and P. In this way the variables that have the most effect on the value of the objective function are targeted first.

A feature which distinguishes branch-and-bound algorithms from one another is the way they select the next node (i.e. subdivision of the solution space) to be solved

from the set of nodes that have yet to be examined. Associated with each node in the search tree is a lower bound on the cost of the primal function within that sub-domain of the solution space (the solution of the relaxed function R over the sub-domain). We follow [Ryoo95] in selecting the node in the search tree with the lowest lower bound to examine in each iteration of the algorithm. This ensures that the global lower bound, L, is improved upon as early on in the search as possible. This is because each iteration focuses on the node in the search tree that represents the global lower bound.

4.4.2 Forming Convex Relaxations

As observed previously, the branch-and-bound algorithm relies on the formulation of easily solvable relaxations of each non-convex component of the original problem. We observe that in our mathematical formulation of the Concave TCFA problem the only non-convex components occur in the objective function, (4.1). In fact the objective function is a separable function with concave components. That is, (4.1) is a summation of functions of a single variable,

$$Z(c) = \sum_{(i,j) \in L} P_{ij}(c_{ij}) \tag{4.16}$$

where

$$P_{ij}(c_{ij}) = \beta_1 c_{ij}^{\alpha_1} + \beta_2 c_{ij}^{\alpha_2} + \left(\beta_3 c_{ij}^{\alpha_3} + \beta_4 c_{ij}^{\alpha_4}\right) l_{ij} \tag{4.17}$$

each of which is concave due to the restriction in the link cost function which ensures that $0 \leq \alpha_{n=1,\ldots,4} \leq 1$. This means a linear underestimating relaxation $R(x)$ of each non-convex component $P(x)$ can be formulated as follows:

$$R(x) = mx + c \tag{4.18}$$

where

$$m = \frac{P(\lceil x \rceil) - P(\lfloor x \rfloor)}{\lceil x \rceil - \lfloor x \rfloor} \tag{4.19}$$

and

$$c = P(\lfloor x \rfloor) - m\lfloor x \rfloor \tag{4.20}$$

$\lceil x \rceil$ denotes the upper boundary of the domain of x and $\lfloor x \rfloor$ the lower boundary.

The resulting relaxed problem is a Mixed Integer, Linear Program (MILP), which we solve using the package LP Solve [Berk95]. The solution to the relaxed problem associated with each node is a guaranteed lower bound of P over the domain over which the relaxed function is formulated.

4.4.3 Improving the Upper Bound

Having formulated and solved a lower bounding problem we employ a local search optimisation technique, Sequential Quadratic Programming (SQP), to attempt to improve on the upper bound on the optimal solution to P in each iteration of the search. See [Borc94] for an example of the use of SQP as a subprocedure of a branch-and-bound algorithm to solve convex minimisation problems.

SQP, described in [Bert95] and [Luen89], is a method for solving general convex nonlinear programming problems and is known to be one of the best methods available for doing so [Bert95][Luen89]. Given a NLP problem of the form:

$$\begin{aligned} &\min && f(x) \\ &\text{subject to:} && g_j(x) \le 0, j = 1, \ldots, r \end{aligned} \tag{4.21}$$

the SQP method attempts to solve the NLP by solving a sequence of quadratic programs with linear constraints. These quadratic sub-problem are of the form:

$$\begin{aligned} &\min && \nabla f(x^k)^T d + \frac{1}{2} d^T H^k d \\ &\text{subject to:} && \nabla g(x^k)^T d + g(x^k) \le 0 \end{aligned} \tag{4.22}$$

where H^k is the Hessian matrix of $f(x^k)$. The solution of the quadratic sub-problem, d^k, is the direction within the solution space in which the search progresses at each iteration of the search. Each new solution is generated according to:

$$x^{k+1} = x^k + \alpha^k d^k \tag{4.23}$$

where α^k is a non-negative, scalar step size. The initial vector $x0$ is arbitrary and the step size, α^k, can be chosen according to one of a variety of rules, the most popular of which is the *Armijo* rule (see [Bert95] for details).

Our code uses a SQP routine called CFSQP [Lawr96]. The CFSQP procedure is started from the solution of the lower bounding problem, R, for the node of the search tree under examination and searches the surface of P for a better feasible solution (i.e. upper bound). Because SQP is a gradient based search procedure designed to optimise convex functions, when it is applied to a non-convex function, such as P, it will most likely become stuck in a local minima of P. Despite this it has the potential to improve on the upper bound of P for a given domain of the solution space and so shorten the global minimisation procedure.

4.4.4 Time and Memory Requirements

Ryoo observes (in [Ryoo95]) that although branch-and-bound algorithms may find the optimal solution to the problem very quickly, they often take a long time to verify its optimality by exhausting the entire search tree. A key contribution of [Ryoo95] is the presentation of four optimality and feasibility tests to tighten variable bounds in each sub-problem encountered. These tests are shown to significantly decrease the time required to solve a variety of problems taken from the domain of chemical process engineering. However, there is a significant difference between the size of the problems dealt with in [Ryoo95] and those that arise in the context of solving a simplified Concave TCFA problem. Ryoo discusses the application of the algorithm on a problem with 200 variables and 200 constraints. The algorithm searched 183 nodes and required 231 CPU seconds (system and user time) on a Sun Microsystems SPARCstation 2™ to execute.

In contrast, a Concave TCFA problem on a complete 5 node network in which all nodes communicate with all others, involves 110 variables and 145 constraints. Applied to 100 randomly generated 5 node networks the algorithm searched an average of 177 nodes (std.dev. 40) and required an average of 7.61 sec. (std.dev. 1.98) on a Pentium™ 166 MHz PC running Linux (approximately 6.4 times faster than a SPARCstation 2™). A 10 node network creates a problem with 945 variables and 145 constraints. Applied to 100 randomly generated 10 node

networks the algorithm always had to be terminated due to memory limitations (the algorithm was terminated if it consumed more than 100 MB of memory). On average the algorithm searched 4,440 nodes (std.dev. 4.6) (the standard deviation is small because the experiments were all terminated due to lack of memory) and required 103 CPU minutes to reach the termination point.

In conclusion, even with the additional tests (optimality and feasibility) it is often not possible to solve the simplified Concave TCFA problem to optimality.

4.5 Concave Branch Elimination (CBE) Procedure

As discussed previously, Kleinrock and Gerla's Concave Branch Elimination (CBE) procedure [Klei76] [Gerl77] is one of the few TCFA algorithms that is appropriate for use with continuous concave link cost functions. The CBE procedure is described in detail in [Klei76] and [Gerl77], we describe it briefly here.

The CBE procedure employs an iterative approach to the solution of a TCFA problem. In each iteration a number of sub-problems are solved. At the lowest level are the *Capacity Assignment (CA)* and *Flow Assignment (FA)* problems.

The FA problem assumes that a link topology, link capacities, link cost function and a matrix of O-D pair traffic requirements are given. The FA problem is to assign traffic flows to minimum delay paths in order to satisfy the traffic requirements without violating link capacities. Using a flow deviation technique from Fratta, Gerla and Kleinrock [Frat72], the optimum solution to the FA problem can be found.

The CA problem assumes that a set of links, the volume of traffic on each link, a link cost function, the total O-D traffic requirements and a maximum allowable link queuing delay, T_{max}, are given. The CA problem is to assign capacities to each link such that the average network queuing delay is less than T_{max} while minimising the

cost of the resulting network. When the link cost function is linear, the optimum capacity for each link is determined by (Equation (5.46) from [Klei76])

$$C_{i \in M} = \lambda_i + \left(\frac{\lambda_i}{\gamma T_{max}}\right) \frac{\sum_{j \in M} \sqrt{\lambda_j d_j}}{\sqrt{\lambda_i d_i}} \tag{4.24}$$

where M is the set of all links, λ_i is the traffic load on link i, γ is the total traffic requirements between all O-D pairs in the network. T_{max} is a maximum allowable delay and d_i is the slope of the link line cost function at C_i. Gerla [Gerl77] states that the CA formula, (4.24), is appropriate only when the link cost functions are linear with respect to capacity. However, Gerla also says that, despite there being no closed form link capacity expression when the link cost function is concave, the CA formula can still be applied iteratively to determine the link capacity. That is, a simple search for the optimal link capacity can be performed, increasing the link capacity in each iteration, until the desired link delay is attained. As we have approximated the link cost function by a double power-law function, which is concave, the iterative approach can be applied in the solution of a Concave TCFA problem.

The CA and FA problems are combined into a *Capacity and Flow Assignment (CFA)* problem where both traffic flows and link capacities must be determined. The CFA problem is solved iteratively by alternating between the CA and FA solution procedures. Despite the fact that the CA and FA problems can individually be solved to optimality, the combined CFA problem cannot. This is due to the non-convex nature of the solution space of the combined problem.

The TCFA procedure (i.e. the CBE procedure) employs the CFA procedure iteratively in an attempt to design the link topology of the network as well. The TCFA procedure relies on the fact that once a link (or sequence of links, i.e., a branch) has been assigned zero capacity, the flow deviation technique will not route traffic onto it, effectively eliminating that branch from the network. The TCFA procedure selects the best solution by starting the CFA procedure from a number (Kleinrock suggests 5 to 10 in [Klei76]) of randomly generated link topologies. For each random topology the CFA procedure is applied a number of times (Kleinrock suggests 20 to 30 times in [Klei76]) to refine the link topology. Each iteration of the CFA

procedure is started with a random traffic flow. The final solution is the best solution found.

In the results presented in Section 4.7, we employ 30 iterations of the CFA procedure, for each of 10 randomly generated link topologies in the TCFA procedure. In addition, we decided to reset the CFA iteration counter to 2 every time a new best solution is encountered. This ensures that each random topology is refined at least 28 times without improvement before the next random topology is generated. Our results have shown that, increasing the number of random flows or topology iterations increases the execution time of the CBE procedure proportionately without significantly improving the quality of the solutions found.

4.6 Concave Link Elimination (CLE) Procedure

Both Gersht and Grout propose a greedy link elimination procedure to solve the Concave TCFA problem in [Gers90] and [Grou87] respectively. As presented neither Gersht's nor Grout's procedures are suitable for solving anything more that small design problems due to their execution time. In Section 2.6 we observed that Gersht's greedy link elimination procedure is more advanced than Grout's, hence we shall not deal with Grout's procedure any further here.

In this section we present an enhanced version of Gersht's procedure, which we call the Concave Link Elimination (CLE) procedure. The advantage of the CLE procedure over previous greedy link elimination procedures is in the techniques used to reduce the execution time of the algorithm. These enhancements allow the algorithm to be employed both, to design much larger networks, and as part of larger network design algorithms (see for example Chapter 5, [Kers91], [Gavi92a], [Stac97c], and [Stac97a]). As our results (see Section 4.7) show, the CLE procedure performs at least as well as the Gersht procedure, and is considerably faster.

The CLE procedure (and Gersht's procedure) starts with all possible links in the network and eliminates subsets of links maximising the reduction in network cost in each iteration while continuing to satisfy any connectivity or reliability constraints. The algorithm terminates when it cannot eliminate any more links without increas-

ing the network cost or violating the capacity, performance, connectivity or reliability constraints imposed on the design.

As is commonly done in network design problems the CLE procedure initially ignores any link capacity allocation constraints (see [Klei76] [Gerl77] [Dutt92]). In effect we assume that the maximum link capacity constraints in the network are sufficiently large to allow all traffic to be routed via the shortest path between O-D pairs. Since link costs are proportional to link length, the optimum routing policy is shortest path routing over the available links in the network. Link capacities can be assigned using either Kleinrock and Gerla's CA procedure for concave link cost functions (see Section 4.5 or [Gerl77]), or Dutta's equivalent link utilisation method (see Section 4.3 or [Dutt92]). For the results in Section 4.7 we use the latter simply because it is computationally slightly less expensive. Our experiments show that the selection of one method or the other has little impact on the cost of the solutions produced. In [Gers90], Gersht uses a constant link utilisation factor (supplied by the user) to assign link capacities. In either case, the link capacity assignment procedure will not assign capacities that exceed the maximum capacity constraint. The delay constraint ensures that the traffic on each link will not exceed the capacity of that link. If the removal of a given link causes the capacity (i.e. delay) constraint on another link to be violated, then the initial link will not be eliminated.

The concave nature of the link cost function implies that an initial solution including all possible links in the network is the maximum upper bound on the optimal cost of the network. The removal of any link (with non-zero capacity), and hence aggregation of traffic onto the remaining links, will always result in a reduction in network cost due to the concavity of the link cost function. Without constraints the algorithm will aggregate all traffic onto the minimum number of links. The result being a set of minimum spanning trees connecting the interacting nodes in the network. As our procedure relies on the concave nature of the link cost function, we call it the *Concave Link Elimination (CLE)* procedure.

In Gersht's procedure only one link is eliminated in each iteration of the algorithm. This means that Gersht's procedure has an average and worst case running time proportional to the number of links in the network. This running time is $O(n^3)$, where n is the number of nodes in the network. This limits the useful-

ness of the algorithm to designing relatively small networks (up to 20 nodes). To determine the single link elimination that will maximise the decrease in network cost, the algorithm must evaluate all possible single link eliminations. The evaluation of each link elimination requires the repeated use of a shortest-path routing algorithm, whose execution time is also dependent on the size of the network. To illustrate, for a fully meshed 10 node network, Gersht's procedure will evaluate 45 potential link eliminations in its first iteration, 44 in its second iteration, 43 in its third iteration, and so on.

Gersht's suggests that the CLE procedure may be accelerated by eliminating more than one link in each iteration of the algorithm. He suggests that a heuristic based on eliminating the m links (where $m \geq 1$ and decreases with "time") that provide the greatest improvement in network cost in each iteration would be appropriate. The elimination of a link will affect the load on other links as traffic is deviated to an alternate path. If the algorithm is to eliminate multiple links simultaneously, care must be taken to ensure that they are isolated from one another. That is, if eliminating link i affects the load on the set of links AE, the algorithm must ensure that none of the links in AE are eliminated along with i. If multiple links are eliminated without regard for how the elimination of each link affects the traffic on the other links being eliminated, there is a danger that links which should play a significant role in the final solution could be eliminated at an earlier stage. Indeed, the network easily becomes disconnected. This is particularly true in the initial stages of the algorithm (when Gersht suggests that most links should be being eliminated simultaneously) when seemingly insignificant low capacity links could be eliminated in favour of high capacity links.

To allow the algorithm to eliminate multiple "unrelated" links (i.e. links whose elimination does not affect each other) simultaneously, we augment the algorithm to evaluate not only the effect of eliminating each link on the cost of the network, but also its effect on the traffic levels of the other links in the network. Using this information, the algorithm can eliminate all links in each iteration of the search that reduce the cost of the network and do not affect the traffic load on each other. Our results show (see Section 4.7) that enabling multiple link eliminations per iteration

significantly reduces the execution time of the algorithm, without significantly reducing the quality of the design produced.

An additional improvement in the speed of the algorithm is obtained by observing that the violation of constraints caused by the removal of a link from a network may indicate a need to retain the link. For example, to protect against any single link failure, the design may be constrained such that each node must have a minimum degree of two. If the elimination of a given link causes that constraint to be violated then we know that link must appear in the final solution, thus it need not be considered for elimination in future iterations of the algorithm.

A detailed description of the algorithm, including our multiple link elimination method, is as follows:

Step 1 (Initialise): The initial network topology is assumed given (in our experiments we have assumed networks are initially fully meshed). Calculate all O-D pair shortest paths, route traffic and assign link capacities. Store the cost of the initial solution as C. Set the set of required links (i.e. those that must occur in the design to satisfy the constraints) to be empty.

Step 2 (Candidate link elimination): Mark all links not in the set of required links as "unevaluated". Select a link, e, from the set of unevaluated links and mark as "evaluated". Remove link e, and recalculate O-D paths as required. Evaluate reliability and performance constraints. If constraints are not satisfied, reinsert link e, restore the network to its state prior to the removal (If constraint violations indicate link e is required in final design add it to the set of required links.) and go to Step 2. If the removal of link e means all constraints are still satisfied go to Step 3.

Step 3 (Evaluation): (The removal of link e did not violate any constraints.) Route traffic and assign link capacities. If the cost of the new solution is less than the current network cost, append the cost of the new solution, link e and the set of links affected by the removal of link e to a set of candidate link eliminations.

Step 4 (Link restoration): Reinsert link e, and restore the network to its state prior to the removal. If there are no more candidate link eliminations to be evaluated go to Step 5, else go to Step 2.

Step 5 (Link elimination): Start with two empty lists, one for the set of links to be permanently eliminated, call this *E*, and another for the links affected by the elimination of the links in list E, call this *AE*. Iterate through the set of candidate link eliminations generated in Step 3 in ascending order of resulting network cost (i.e. start with the link eliminations which improve the cost of the network most). For each candidate link elimination, if neither the link to be eliminated nor any of the links its elimination affects are included in either *E* or *AE*, add the candidate link to *E* and the set of links affected by its elimination to *AE*. When the list of improving candidate link eliminations is exhausted, set *E* to empty (there are no links to eliminate) and go to Step 6. Otherwise, eliminate the links in *E*, recalculate all O-D pair shortest paths, route traffic, assign link capacities and calculate the cost of the new solution, *C*. Go to Step 2.

Step 6 (Finished - restore best solution): Route traffic, assign link capacities and calculate solution cost, *C*.

In Step 5 of the CLE algorithm all O-D pair shortest paths are recalculated once the link elimination phase has been completed. This is in contrast to the recalculation of only the affected shortest paths in Steps 2 and 4. This raises the possibility that the execution time of the algorithm could be reduced by doing the same in Step 6.

Our experiments have shown that recalculating O-D paths only as required in Step 6 provides no reduction in execution time. The reason for this relates to our use of Dijkstra's algorithm to calculate the shortest paths from each node to all other nodes. The elimination of any link in any path from a given node to any other means that the paths for that node must be recalculated. In each iteration of the algorithm the links eliminated are distributed about the network and will affect most of the network. Our experiments show that in most iterations, the majority of the shortest paths must be recalculated. In addition, the overhead associated with determining which paths must be recalculated tends to outweigh any advantage gained. In the final analysis the best performance is gained by simply recalculating all O-D pair shortest paths at the end of each iteration of the algorithm.

4.7 CLE vs. Gersht vs. CBE vs. Lower Bound

We evaluate the performance of the CBE, Gersht and CLE procedures by applying them to a number of randomly generated network design problems of varying sizes. The Gersht procedure employed to obtain these results includes the addition of our proposed enhancement to identify links that must be included in the final design. That is, the algorithm does not evaluate links for removal that have previously been identified as required in the final solution. The algorithm is, however, limited to eliminating only one link per iteration, as in Gersht's original description of the greedy link elimination algorithm in [Gers90]. The cost of the solutions produced by each procedure is also compared with the bounds produced by the branch-and-bound procedure for the same set of networks.

4.7.1 Random Network Design Problem Generation

We generate the random network design problems by randomly distributing N nodes (where N is the size of the network) about a 1,000 km^2 area. Any link is allowed in the design so the initial topology is a full mesh over the N nodes.

Most randomly generated design problems in the literature assume that all node pairs have symmetric, uniform traffic requirement (e.g. [Klei76], [Gerl77], [Pirk88], [Gavi89]). This leads to balanced network designs which are often not particularly realistic [Dutt92]. A more realistic method of generating a random traffic requirements matrix is employed in [Dutt92] where O-D pair traffic requirements are sampled from a normal distribution (bound below by zero). Although better, the traffic requirements produced using this method are still not particularly realistic as it assumes the volume of traffic between node pairs to be independent of the client population at each node and the distance between them. We observe that the traffic requirements between two nodes are often proportional to the populations (user or computer) located at each node and inversely proportional to the distance between them.

To generate a realistic "random" traffic requirements matrix we first select a "user" population for each node from a Pareto distribution (Equation (4.25)) which is commonly used to model population sizes [Devo82].

$$f(x) = \begin{cases} \dfrac{k\Theta^k}{x^{(k+1)}} & , x \geq 0 \\ 0 & , x < 0 \end{cases} \qquad (4.25)$$

We assume values for $\Theta = 1,000$ and $k = 0.8$ to produce an exponentially decaying population size distribution with a long tail and a lower bound of 1,000. Varying the values of these parameters will alter the population sizes and hence the volume of traffic in the network. Our experiments show that the impact on the relative performance of the solution procedures due to varying these parameters is insignificant.

In order to translate a node population size into a traffic volume we assume that each node represents a community such as a university campus and generates wide area network traffic accordingly. We recorded the mean volume of Internet traffic on the link connecting the University of Wollongong to the rest of the Internet over a 36 week period. During this time the link carried an average of 18 GigaBytes per week in each direction. Converting to Megabits per second, assuming a user population at the university of 10,000, with 5 active days per week and 10 active hours per day, produces an estimate of the mean volume of traffic generated (and received) per user per hour during a busy hour, $K = 8.263 \times 10^{-5}$ Mb/s/user. The actual value of K is not critical, it simply allows us to translate a client population into a traffic requirements distribution. Our experiments have shown that altering K alters the volume of traffic flowing in the network and does not significantly affect the relative performance of the solution procedures examined here.

The resulting traffic required to flow from node i to j, denoted γ_{ij}, in Mb/s, is given by:

$$\gamma_{ij} = \frac{KP_i P_j}{d(i,j) \sum_{k \in N-i} P_k} \quad (4.26)$$

where P_i is the population at node i, and $d(i,j)$ is the geographic distance between nodes i and j. The traffic generated at node i, KP_i, is distributed to the other nodes in the network in proportion to the population located at each other node (as a percentage of the total network population, less the population at node i), $P_j/(\sum_{k \in N-i} P_k)$. The distance between nodes is introduced to reduce the interaction between nodes as the distance between them increases (in this context, distance is considered to be a dimensionless, scaling factor to keep the expression dimensionally sound). The assumption in this context is that we are designing networks to support services in which the interaction between nodes decreases with distance. If this is not appropriate the distance term can be removed without altering the effectiveness of the CLE procedure.

4.7.2 Solution Cost and Execution Time Results

Figure 4.2 shows the relative cost of the solutions generated by the CBE, Gersht, and CLE procedures. All solution costs have been normalised with respect to the cost of the solution found by the CLE procedure to allow the aggregation of results from designing 100 randomly generated networks of 5 through 20 nodes and 50 of 30 though 50 nodes. The error-bars represent the 90% confidence intervals for each estimate.

For small networks all three procedures produce very similar results. The Gersht procedure has a slight (less than 1%) advantage over the CLE procedure (the results for the Gersht procedure are truncated due to the execution time of the branch-and-bound procedure for larger networks). As the network size increases the CBE procedure shows a marked degradation in performance relative to the CLE and Gersht procedures. The difference between the cost of the solutions produced by the CBE and CLE procedures increases until the network size reaches approximately 20

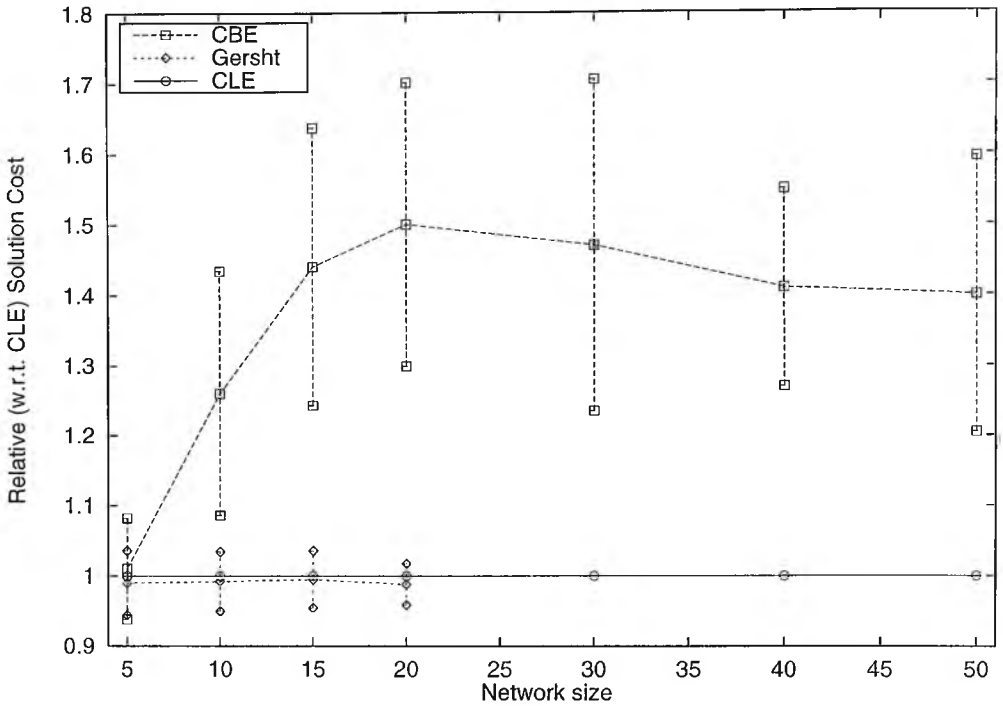


Figure 4.2 CBE vs. Gersht vs. CLE: Solution cost comparison.

nodes at which point the difference is approximately 45%. The difference then decreases slightly and plateaus at approximately 40%. The performance of the CBE procedure degrades as the size of the solution space increases, which increases with the size of the network. In a larger solution space there are more local minima for the CBE procedure to become stuck in, so its relative performance degrades. The degradation in performance is balanced by the effect of the general shape of the solution space. Numerical results have shown that the solution spaces of large scale network design problems are often relatively flat near the global minima [Gerl77] [Gers90]. This means that although there are many local minima in the solution space many of them lie at the same level near the global minima. Thus, the fact that the CBE procedure gets stuck in local minima is mitigated by the shape of the solution space which causes the plateau effect seen in Figure 4.2.

Figure 4.3 shows the results of comparing the CBE, Gersht and CLE procedures with the bounds produced by the continuous branch-and-bound procedure, for 5 and 10 node networks. As before, all solution costs have been normalised with respect to the cost of the solution found by the CLE procedure to allow the aggregation of results from designing 100 randomly generated networks. The error-bars represent the 90% confidence intervals for each estimate.

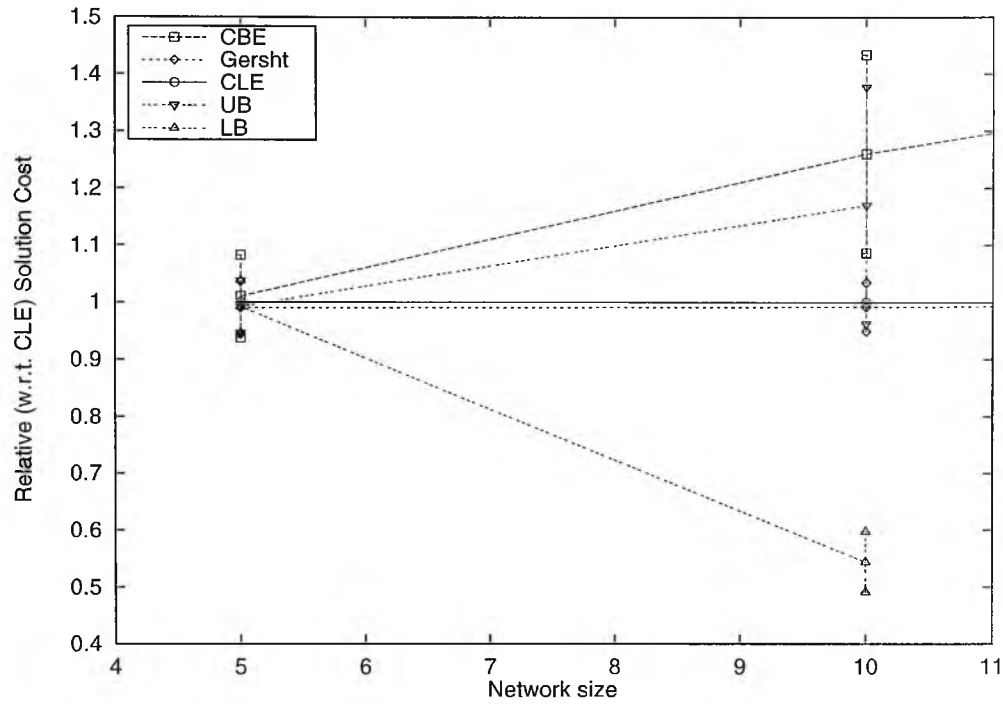


Figure 4.3 CBE vs. Gersht vs. CLE vs. Bounds: Solution cost comparison.

For small networks (5 nodes) all three procedures perform well, frequently finding the optimum solution. For small networks the gap between the upper and lower bounds produced by the branch-and-bound procedure is small. This indicates that the procedure was able to solve the simplified Concave TCFA problem to optimality. The fact that the estimates of the average performance of all three procedures are within or very close to the bounds indicates that they were frequently able to find the optimal solution.

The gap between the bounds on the cost of the optimal solution produced by the branch-and-bound procedure rapidly increases as the size of the network increases. This is due to the fact that the procedure is unable to solve the problem to optimality within reasonable time or memory constraints. It is interesting to note that while the CLE and Gersht procedures remain within the bounds for 10 node networks, the CBE procedure is already outside the upper bound. This indicates that the CLE and Gersht procedures are quite likely to be finding solutions near the optimum (the optimum is guaranteed to lie somewhere between the CLE/Gersht result and the lower bound), while the CBE procedure is producing results some distance from the optimum.

In these results the design procedures are solving the simplified Concave TCFA problem, from Section 4.3, in which link capacities are assumed to be continuous. However, the procedures are all capable of solving the unsimplified version, from Section 4.2, in which link capacities are selected from a discrete set of allowable capacities, or occur in discrete increments. Our experiments show that the use of discrete link capacities has little effect on the results.

Figure 4.4 shows the average execution time, in CPU seconds on a Sun Microsystems SPARCstation 5™, of the CBE, Gersht and CLE procedures as the network size increases. Again, the estimates are taken from 100 randomly generated networks of 5 through 20 node and 50 of 30 though 50 nodes. Similarly, the error-bars indicate the 90% confidence interval for each estimate.

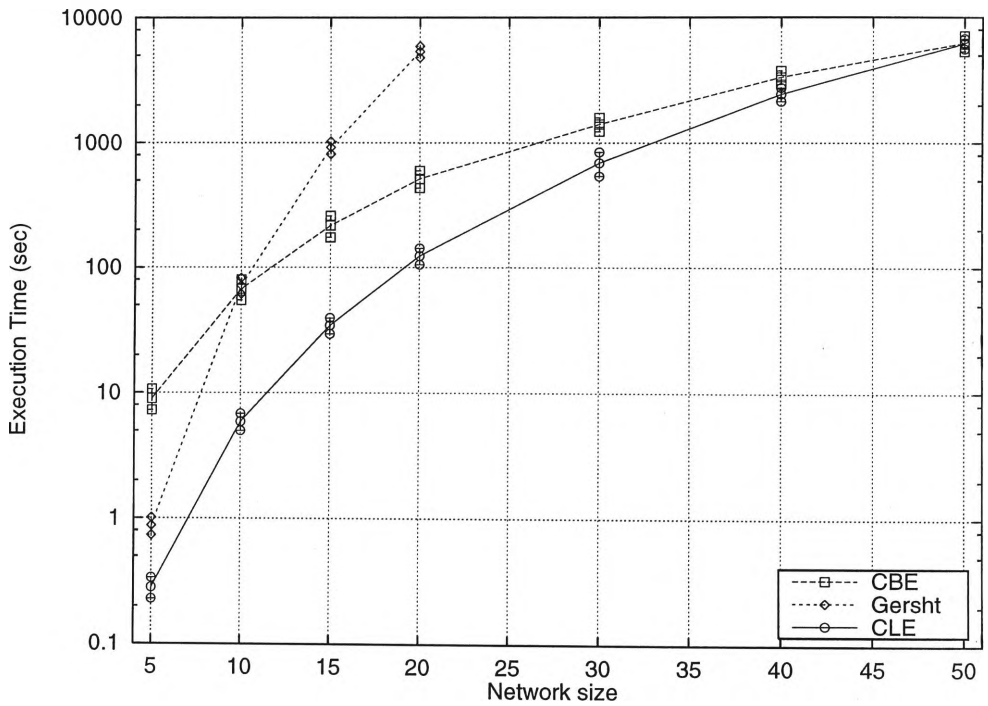


Figure 4.4 CBE vs. Gersht vs. CLE: Execution time comparison.

Although the worst case running time of the CLE procedure is still $O(n^3)$, Figure 4.4 shows that its average case performance is significantly better. The CLE procedure is consistently faster than both the CBE and Gersht procedures. For small networks (5 nodes) the CLE procedure is an order of magnitude faster than the CBE procedure. As the network size increases the difference between the CBE and CLE procedures decreases until the time required to design a 50 node network is comparable. In contrast, the difference in execution time between the CLE and Gersht proce-

dures increases with the network size. At networks of 11 nodes or more, the Gersht procedure takes longer than either the CLE or CBE procedures. Our experiments indicate that the design of a 30 node network using the Gersht procedure takes in the order of 27 hours. In contrast, the CLE procedure takes approximately 15 minutes to design the same network. This is to be expected as the Gersht procedure's execution time is strongly related to the number of links in the network (i.e. its average and worst case running time is $O(n^3)$).

4.8 Conclusions

This chapter shows that of the three design procedures presented to solve the TCFA problem, only our CLE procedure is appropriate for use in the presence of concave link cost functions. Both Gersht's greedy link elimination procedure [Gers90] and the CLE procedure produce consistently cheaper network designs than Kleinrock and Gerla's CBE [Klei76] [Gerl77]. However, Gersht's procedure is practical for small networks only (i.e. up to 20 nodes) due to it's execution time. Our results show that our CLE procedure produces network designs whose cost is within 1% (on average) of those produced by Gersht's procedure, in significantly less time (the CLE procedure is almost two orders of magnitude faster than Gersht's procedure in designing 20 node networks). When compared to the CBE procedure the cost of the designs produced by the CLE procedure are up to 45% (on average) cheaper, and again take less time to produce.

To provide a quantitative assessment of the performance of the design procedures we formulated the Concave TCFA problem as a MINLP in Section 4.2. The problem was relaxed in Section 4.3 to form a lower bounding problem which we solved using a continuous branch-and-bound algorithm. The results obtained by comparing the design and bounding procedure when applied to a large number of randomly generated networks show that for small networks (up to 10 nodes) the CLE procedure is shown to be producing solutions whose cost is, on average, at most 45% from the lower bound (which we know to be loose).

These results lead us to employ the CLE procedure in the larger process of designing a DSN. We examine that process in more detail in the next chapter.

5. Distributed Server Network Design Procedures

If it is fast and ugly, they will use it and curse you; if it is slow, they will not use it.

- David Cheriton

5.1 Introduction

In Chapter 3 we observed that the design of DSN's can be broken into two subproblems. These are: finding the optimal configuration of servers, and the design of the network connecting client nodes to servers and servers to one another. Recall that a configuration of servers defines the number and location of servers and the assignment of client nodes to them. This information is sufficient to determine the traffic requirements between nodes. In Chapter 4 we developed the CLE procedure to design a network given a traffic requirements matrix. In the first part of this chapter (Sections 5.2 and 5.3) we develop three heuristic design procedures to determine the optimal configuration of servers in the network. One is based on node clustering, while the other two are based on ADD/DROP and ADD-k greedy searches respectively. An analysis of the complexity and execution time of these heuristic design procedures is presented in Section 5.4.

In Section 5.5 we examine methods which allow us to assess the performance of the heuristic design procedures. Ideally we would like to find the optimal solution to the original problem. However, the design of a DSN is an NP-hard, combinatorial optimisation problem. This makes finding the optimal solution (via an exhaustive search) infeasible for all but small problem instances. For larger problems we deter-

mine a lower bound on the cost of the optimal solution by formulating and solving a less complex lower bounding problem.

5.2 Node Clustering Procedure

Node clustering techniques are intuitively appealing in the context of the DSNDP. In the absence of complicating constraints we expect to see clusters of nodes appearing in the optimal solution, with each cluster of nodes being assigned to a single server within the cluster.

Node clustering algorithms are employed in a number of disciplines to identify subsets of nodes that share some common feature and so should be grouped/clustered together. Nodes may represent pixels in an image [Cole79], bodies of ore, words in a natural language database, or the location of populations of clients in a communications network.

In Section 5.2.1 we describe the general operation of the majority of the node clustering algorithms. Section 5.2.2 reviews some of the node clustering algorithms that have been applied to network design problems in the literature. A node clustering algorithm that we have developed to aid in the design of DSN's is presented in Section 5.2.3.

5.2.1 General Operation of Clustering Algorithms

Clustering procedures typically operate by clustering nodes into disjoint sets of nodes based on some relationship between them. Ideally, each node should be more strongly related to the nodes in its own cluster than the other nodes in the network.

Clustering algorithms typically follow one of two approaches to determine the number of clusters to be formed.

The first approach allows the cluster formation procedure to determine the number of clusters automatically. Algorithms that follow this approach usually use a gravity style model to group nodes together. A "nearness" [Klei80], or "pull" [Saha95], function is defined to capture the relationship between node pairs. Nodes with a strong attraction to one another will usually appear in the same cluster. Cluster for-

mation is performed by starting with all nodes in clusters by themselves. The two clusters with the strongest attraction to one another are then joined to form a single cluster. The new cluster is treated as a single node located at the "centre of mass" of its constituent nodes, with a "mass", or attractive force, equal to the sum, or average, of its constituent nodes. This process is repeated until no more clusters can be joined. A limit is usually placed on the size of clusters to prevent the nodes collapsing into one large cluster.

The main disadvantage of gravity based clustering algorithms in the context of DSN design is the need to place an arbitrary limit on the maximum size of clusters. The size of a cluster (and hence the capacity of its associated server) should be determined by the interactions of the cost functions, the distribution of nodes and their interaction. It should not however be based on a limit introduced to prevent the nodes from collapsing into a single large cluster.

The second approach is to form a specified number of clusters. A meta-heuristic can be used to determine the optimal number of clusters and hence clustering. A simple (and commonly employed) method is to iterate through all possible numbers of clusters. Given the desired number of clusters, C, the algorithm typically starts by selecting C root nodes, each of which represents the "seed" or beginnings of a cluster. The remaining nodes are assigned to each cluster using a "pull" function. The pull function is designed such that clusters formed will optimise the value of an "energy" function, which is usually simply a summation of the "pull" function over all pairs of nodes. The energy function is used to evaluate a given set of clusters.

The second approach to clustering has the advantage that cluster size determination is determined by the algorithm itself. If there are hard limits on the size of a cluster then they can easily be incorporated into the algorithm.

In the context of DSN design there are two additional steps required to transform a clustering of nodes into a DSN. The first is the location of a server within each cluster (we assume that all nodes in a cluster are assigned to the same server located within that cluster). The second is the design of the intra and inter cluster network topologies, that is the network connecting clients to their servers and servers (clusters) to one another.

5.2.2 Node Clustering in the Network Design Literature

Gravity based node clustering algorithms are employed in [McGr77], [Diri77] and [Schn82] to determine the location of concentrators, or access facilities, in each level of a hierarchical network. The objective, in each case, is to connect remote terminals to a central site.

In [Klei80], Kleinrock examines the use of clustering algorithms to reduce the complexity of designing large hierarchical networks. Node clustering is used to break the design of a large network into a number of smaller problems, each of which can be handled by existing network design procedures, such as the CBE procedure. Kleinrock aims to cluster nodes and determine the number of levels in the hierarchy to minimise the computational complexity of the overall design. This now seems an unusual goal, given that we usually wish to minimise the cost of the network design rather than the cost of designing it. However, the speed and cost of computers at the time may well have had an influence. Kleinrock does not present a node clustering algorithm, instead he develops expressions which determine the optimal cluster size (given the number of nodes and levels in the network), or determine the optimal number of levels, given a uniform cluster size (and topology).

As discussed in Chapter 3, Saha *et al.*, present a clustering algorithm to aid the design of a two level communications network in the presence of node and link failures in [Mukh93a], [Mukh93b] and [Saha95]. As observed in Section 3.3.2, despite the contradictions in the presentation of their node clustering algorithm, it offers a powerful way to identify promising configurations of backbone nodes and assignment of user nodes to them. The power of the clustering algorithm comes from the use of a "pull" or "gravity" style function to identify relationships between nodes. In addition, the algorithm does not rely on any arbitrary user input to guide it to a solution.

A portion of the hierarchical network design problem is dealt with in [Shar93], where the O-D pair traffic requirements, a set of clusters, inter and intra-cluster topologies, cost and link delay constraints, are given. [Shar93] attempts to: (i) identify the inter-cluster topology and link capacities, and (ii) select gateways in each cluster. The aim is to maximise the network reliability to cost ratio. Gateways within each cluster are selected by evaluating all possible combinations of 1 to n (where n

is the number of nodes in the cluster) combinations of gateways. The combination that optimises the objective function is chosen as the solution.

A clustering algorithm developed to solve a variation of a simplified version of the DSNDP is presented in [Monm86] (a design tool based on [Monm86] is described in [Card89]). The key feature of interest is that user nodes are allowed to belong to more than one cluster. The clustering algorithm is used to design a backbone network. Nodes in the backbone network are either access facilities or switches. Low volume users are attached to access facilities, while high volume users (e.g. gateways to other networks, databases, etc.) are attached directly to switches. Switches are assumed to be fully meshed. User nodes (including access facilities) are assumed to want to home to: one switch, all switches or some subset of switches. The design procedure starts by estimating the number of switches required, S. Then S clusters of user nodes (including access facilities) are formed. Users that have to be homed to all switches are automatically members of all clusters. In addition, user nodes with a high volume of traffic start off being assigned to all clusters. The remaining nodes are assigned (in order of descending throughput) to the cluster that maximises a "goodness of fit" measure for that node, cluster pair. Once all nodes have been assigned, the clustering is perturbed to ensure that all nodes are correctly assigned and those that are to be assigned to a "subset" of switches are assigned correctly.

While the clustering procedure from Saha *et al.*, presented in [Mukh93a], [Mukh93b] and [Saha95], is promising it is deficient in a number of ways when applied to a DSNDP. For example, Saha *et al.* do not include any aspects of network topology design in their work. They measure the distance between nodes geographically, rather than considering the wire-line distance between nodes which may be considerably larger in some cases. Nor do they include the design and hence cost of the network required to connect their nodes and clusters together to assess the quality of a given clustered arrangement of nodes. They consider the network only when the clustering of nodes has been decided and then they consider the backbone network only. Saha *et al.* also assume that all nodes are "equal" rather than considering that some nodes may contribute a much larger proportion of traffic in the network and justify altering the design accordingly. In addition, all nodes are considered as potential server locations, something which may not possible due to external

resource or politic constraints. Moreover, when selecting gateways (or servers) for each cluster Saha *et al.* always select the node closest to the centre of the entire network. This will often produce very unbalanced intra-cluster network topologies, which are likely to be sub-optimal. In the next section we propose our own node clustering procedure, based largely on that of Saha *et al.*, but enhanced to address the deficiencies identified such that it may be applied successfully to the DSNDP.

5.2.3 Proposed Node Clustering Procedure

The clustering procedure we propose follows [Mukh93a] and [Saha95] in employing the second approach to node clustering described in Section 5.2.2. That is, a procedure to form a given number of clusters is wrapped inside a loop which determines the optimum number of clusters to use. We chose to base our algorithm on [Mukh93a] and [Saha95] for a variety of reasons. Firstly, as mentioned, Saha *et al.* [Mukh93a] [Saha95] consider the design of a two-level network, similar to the DSN structure. The iterative nature of the algorithm does not place any arbitrary limits on the size or number of clusters. The gravity based "pull" function used in this type of algorithm is easily modified to capture the relationship between nodes. Due to the two-level network design problem that Saha *et al.* consider, they include the task of selecting gateway nodes within each to cluster to interconnect the clusters. The relatively modular form of the algorithm means that it can easily be adapted, to either the specific problem at hand, or to examine the merits of different approaches to solving the problem. Finally, a relativity complete set of example inputs to their algorithm and its subsequent output are provided in [Saha95] making it possible to reproduce their results as a starting point for our own investigations.

Following [Mukh93a] and [Saha95] the output of our clustering algorithm is a set of clusters of nodes, with one node in each cluster designated as the location of the server to which all nodes in the cluster are assigned. In addition to identifying clusters of nodes, our algorithm also determines the topology and capacity of links connecting client nodes to their server within each cluster and clusters to one another. This differs from [Mukh93a] and [Saha95], where the network link topology design is not included.

The clustering algorithm uses a "pull" function to capture the relationship between nodes. We have assumed that the traffic generated by each client node is proportional to the size of its associated user population. Hence, we consider the size of the user population at a node to be equivalent to the pull (or force) it exerts on other nodes. That is, a node's population is analogous to its mass in a gravitational sense. Similarly, the traffic volume generated at a node is also analogous to the mass of the node. We also assume that the attraction between nodes is inversely proportional to the square of the distance between them. Similar pull functions are employed in [McGr77], [Diri77], [Schn82], [Klei80], [Mukh93a] and [Saha95], however, none of them take the volume of traffic generated by nodes into account, thus implying that all nodes are considered equal (i.e. all have unit mass). Our directed "pull" function has the form:

$$p(u, v) = \frac{\text{mass}(u)}{d(u, v)^2} \tag{5.1}$$

where $d(u, v)$ is the link distance between nodes u and v, and mass(u) is the size of the population associated with node u. $p(u, v)$ is effectively the force exerted on node v by u. In a DSN, the volume of traffic between a client and server depends on the size of the population at the client node only. Hence, we include the "mass" of only the client node in the pull function.

Our formulation of the DSNDP does not include the existence of background (peer-to-peer) traffic due to other services. However, Equation (5.1) is easily modified to capture the impact of background traffic. Including background traffic in the network would mean that the cost of transporting a given volume of traffic over a link would depend not only on the capacity requested and length of the link, but also on the volume of traffic already present on the link (i.e the volume of background traffic). If the link cost per unit capacity varies across links, due to either the existence of background traffic, or varying cost function parameters across links, then the cost of transporting an arbitrary volume of traffic between the nodes of interest could be used in Equation (5.1), instead of the link distance between them.

To form a set of C clusters, the algorithm first selects C "root" nodes around which the clusters are built. In the context of a DSN we must ensure that each cluster includes at least one potential server location, so only potential server locations are

considered as potential root nodes. We have investigated two methods for selecting root nodes.

The first root node selection strategy is based solely on minimising the "pull" between root nodes. The first two root nodes are the pair of potential server locations which have the weakest attraction to one another (as determined by the pull function) (i.e. the two potential server locations farthest apart). Each additional root node is found by selecting the free potential server location that has the minimum mean attraction toward the current set of root nodes (i.e. is farthest from all other root nodes). This node is added to the set of root nodes, and the process repeated until the desired number of root nodes is found. This strategy is similar to that employed in [Mukh93a] and [Saha95]. The main difference is that they consider all nodes to be potential server locations.

The second strategy that we propose biases the selection of root nodes towards potential server locations at "centres of mass" in the network. A centre of mass (CoM) measure, CoM(u), is calculated for each potential server location, u.

$$\mathrm{CoM}(u) = \sum_{v \in N} p(u, v) \tag{5.2}$$

where N is the set of all nodes. Root nodes are conditionally selected in descending order of their CoM measure. Before a node, u, is selected as a root node a check is performed to determine if the choice is justified. If placing a server at u reduces the cost of the network with respect to the existing set of root nodes, then node u is selected as a root node. If the algorithm determines that many of the potential server locations do not justify being used as root nodes, it is possible that the list of potential server locations may be exhausted before the required number of root nodes has been selected. In this case, the remaining root nodes are selected from the remaining potential server locations, so that the pull between them and the existing set of root nodes is minimised (i.e. the strategy above).

Once the root nodes have been selected, clusters are built around them using the cluster formation algorithm presented in [Mukh93a] and [Saha95]. Associated with each permanent cluster is a temporary cluster which stores nodes before they are assigned to a permanent cluster. For each free node (i.e. one that has not been assigned to a cluster), the average pull exerted on it by the nodes in each of the

existing clusters is calculated. Each free node is assigned to the temporary cluster corresponding to the permanent cluster that it had the greatest attraction to. Once all free nodes have been assigned to temporary clusters, the node in each temporary cluster that has the greatest average attraction to the associated permanent cluster, is added to the permanent cluster, and removed from the set of free nodes. At most one node is added to each cluster in each iteration. All nodes remaining in the temporary clusters are declared free again, and the process repeats until all nodes are assigned to a permanent cluster.

For any given number of clusters, C, and set of potential server locations, nodes will be formed into a unique set of clusters. This is because the formation of these C clusters is completely determined by the node populations and distances between nodes.

Having formed the required number of clusters, the volume of traffic delivered to each server and amongst servers can be determined. Note that server capacities are not dependent on their location, only on the nodes assigned to each server and the number of servers in the network.

The next step in the clustering design algorithm is the selection of the location of a server in each cluster. The root node selection process guarantees that there will be at least one potential server location in each cluster, that is the root node of the cluster. If there is only one potential server location in a cluster, then it becomes the location of the server for that cluster. If however, there is more than one potential server location in a cluster, a choice must be made. We investigated two strategies for selecting the server location in each cluster.

In the first strategy, the potential server location closest to the CoM (or "wire centre") of each cluster is chosen as the server for that cluster. In this case the CoM of a cluster of nodes is defined as that node which minimises the sum of the weighted (by node population) link distances between it and all other nodes in the cluster. The CoM measure employed in this instance is similar to the CoM measure used in the root node selection strategy. It differs in that it uses the link distance rather than the link distance squared. This is because all traffic generated by each node will be delivered to the server, and hence the volume of traffic will not decrease with distance. An additional, dummy node, is added to each cluster when calculating the

CoM measure in this context, to capture the effect of the inter-cluster (i.e. inter-server) traffic generated by a cluster. The node at the CoM of the entire network is used as the location of the dummy node. Its mass is set to be equivalent to the size of a population that would generate the same volume of client-server traffic as the volume of inter-server traffic generated by the cluster. A CoM based approach to the location of servers, or gateways, is also followed in [McGr77], [Diri77], [Schn82], and [Klei80]. This strategy, while fast, may select poor server locations if the topology of the cluster is particularly unbalanced.

In the second strategy, the server location that minimises the cost of the intra-cluster network is selected. Each potential server location is evaluated by designing the intra-cluster network required to connect all nodes to that location. A dummy node is added to the cluster as before to capture the influence of inter-server traffic. The intra-cluster network topology is designed using the CLE procedure from Section 4.6. A search based approach to the location of servers, or gateways, is also followed in [Shar93]. This strategy is slow compared with the CoM based strategy. However, it will not be adversely affected by unbalanced cluster topologies. We discuss these two strategies further in the next section and compare their relative performance in Section 6.4.1.

Once the clusters and server locations are determined, the volume of traffic delivered to each server and amongst servers can be determined. This information is used to design the network link topology in either of two ways. Either individual intra and inter-cluster traffic requirements matrices may be generated and each portion of the network designed separately. Alternately, a single requirements matrix can be generated for the entire network, which is then designed as a whole. In either case the CLE procedure is used to design the required link topologies.

Designing the network as a whole allows the intra and inter-cluster traffic to be combined, thus taking advantage of the concave link cost function. The disadvantage of this approach is the complexity of designing a large network. Designing the network in portions overcomes this difficulty, as designing a number of small networks is faster than designing a single large network. The disadvantage of piecemeal network design is that the aggregation of intra and inter-server traffic will not occur. However, in some environments this may in fact be desirable, or even neces-

sary, as the intra and inter-cluster traffic may be assigned to different virtual layers in the network. In an effort to take advantage of the execution time advantage of the piecemeal design approach, while maintaining the advantage of aggregating traffic we examine the effect of combining both methods. That is, the algorithm may use the piecemeal strategy while determining the optimal number of clusters, but once that is determined the network is redesigned as a whole.

The cost of the network designed with C clusters represents the value of the objective function for that potential DSNDP solution. To find the optimum solution, the algorithm iterates on the number of clusters formed from 1 to P, the number of potential server locations (we note that Saha *et. al.*'s design procedure iterates from 2 to N clusters, thus excluding the possibility of employing only one server in the network). The optimal number of clusters (and hence the optimal solution) is that which minimises the total cost of the network.

5.2.4 Clustering Algorithm Variations

As described above, the node clustering based DSN design procedure comprises several parts. For some of these parts, we have outlined two alternative implementations. Here we examine the performance of various combinations of these implementation choices, to determine the best combination. Before describing these combinations, we summarise the different options.

The first implementation options relate to the form of the "pull" function (Equation (5.1)) used to measure the strength of the association between nodes. In contrast to previous work (e.g. [Mukh93a] and [Saha95]) we suggested that the "mass" of clients nodes (i.e. the size of their associated population) be included in the "pull" function. So the first implementation decision is whether node mass should be included in the "pull" function.

The second implementation decision determines which of the two "root" node selection strategies described in Section 5.2.3 is used. One option is to use the simple strategy that selects a set of nodes that maximises the distance between them. The second option is to use the more complex procedure that selects the nodes with the largest client populations, but checks to make sure that each root node added is

justified given the set of nodes already selected. If this method runs out of nodes it is able to select the additional root nodes required using the simple strategy.

The third implementation decision determines the location of the server within each cluster of nodes. One option uses an exhaustive search over all possible potential server locations in each cluster. The second, faster, option is to locate a server at the Centre of Mass (CoM) of each cluster.

The final implementation decision determines whether the link topology of the network is designed as a whole, or in parts. If the link topology is designed as a whole, the CLE procedure has to solve a larger problem, but may take advantage of the concave link cost function by co-locating intra and inter-cluster traffic flows. The alternative is to treat the design of the link topologies within each cluster and between clusters as separate problems, the solutions to which are combined to provide the complete network topology.

Although, the operation decisions described above produce sixteen possible combinations, we have chosen to examine and present only seven in detail. Our choice of combinations reflects our attempts to reduce the execution time of the procedure without adversely affecting solution quality. The combinations of operational modes not presented were discarded after initial experiments indicated they did not provide any improvement over the combinations already chosen. Table 5.1 shows the variations of the node clustering based DSN procedure examined. Note that the "Clustering Procedure" label is simply an arbitrary label used to identify each variation of the procedure. The "fast" and "slow" clustering procedures in our results correspond to procedures **A** and **G** respectively.

A comparison of the various forms of the clustering algorithm is presented in Section 6.4.1. The next section discusses a variety of "greedy" local search based design procedures that may be used to solve the DSNDP as an alternative to our clustering procedure described above.

Table 5.1 Node clustering based DSN design procedure variations.

Clustering Procedure	Include node mass in "pull" function?	Use Simple, or Complex root node selection?	Use search, or CoM based server location?	Link topology designed as a Whole, or in Parts?
A	Yes	Complex	Search	Whole
B	Yes	Simple	Search	Whole
C	No	Simple	Search	Whole
D	Yes	Simple	CoM	Whole
E	Yes	Simple	Search	Parts
F	Yes	Simple	CoM	Parts
G	Yes	Complex	CoM	Parts

5.3 Greedy Local Search Procedures

The greedy search heuristics are often used in optimisation, because of their simplicity and robustness. A greedy search algorithm makes no assumptions about the form of the optimal solution, it simply moves through the solution space in the direction of maximum improvement in solution cost. Greedy algorithms attempt to improve on the current solution by generating a set of potential next step solutions using a solution generation heuristic, such as the ADD/DROP or ADD-*k* heuristics described below. Each potential next step solution is evaluated and the one that maximises the improvement in cost is chosen to be used as the current solution in the next iteration of the search. The search finishes when no better solution can be found.

We apply two greedy search heuristics, each of which searches the solution space of possible configurations of servers. For any given set of servers, a solution is generated by homing client nodes to their nearest server based on link distance.

While the idea of clusters of nodes is not explicitly identified by the greedy search procedures, they still produce potential solutions involving clusters of nodes. A cluster of nodes is defined as the set of nodes assigned to a single server. While the clustering procedure attempts to identify clusters on the basis of the relationship between nodes, the greedy search procedures simply identify server locations to which nodes are assigned.

Once a set of servers and the assignment of client nodes to them has been determined, the volume of traffic flowing between nodes can be determined. This information allows us to calculate the required capacity of the servers chosen and generate a two-dimensional, O-D pair, traffic requirements matrix for the network. The traffic matrix is used to determine the topology and capacity of links in the intra and inter-cluster portions of the network (either as a whole, or in portions) using the CLE procedure. Like the clustering heuristics, the objective of the search heuristics is to find the set of servers (i.e. solution) that minimises the total cost of the network.

The greedy search procedures can be summarised as follows:

Step 1 (Initialise): Generate an initial solution and record its cost, *C*.

Step 2 (Search for next step): Using an appropriate heuristic, generate and evaluate a set of potential next step solutions from the current solution.

Step 3 (Greedy improvement): If any of the solutions generated in Step 2 represent an improvement in solution cost, select the one that maximises the improvement in solution cost. Make the selected solution the current solution and go to Step 2. If no improving solution was found, the search is finished. The current solution is the best solution found.

5.3.1 ADD/DROP Heuristic

Given a set of servers and potential server locations, an ADD/DROP heuristic can be used to generate a set of potential next step solutions. The ADD/DROP heuristic generates solutions by adding or deleting a single server to/from the current solution.

Unless the solution space is convex (i.e. it has only one minima) local search heuristics, such as a greedy search, are not guaranteed to find the global optimum solution. Instead, they may become trapped in a local minima. To overcome this problem local search heuristics are usually performed a number of times, each starting from a different point in the solution space. The ADD/DROP version of the greedy search procedure starts from either the maximum or minimum number of servers allowed. The algorithm then generates and evaluates all possible combinations of the initial

number of servers from the number of potential servers. The combination that produces the lowest cost network is then used as the starting point for the greedy search. In practice, we have found that starting from the minimum number of servers allowed gives the best results in most circumstances. The only time that this will not be the case is when servers are relatively cheap in comparison to links, and thus the solution space contains ridges. If servers are cheap relative to links, the optimal solution will be weighted towards having a large number of servers. This in itself will not prevent the ADD/DROP heuristic from finding a (near) optimal solution. However, if there are ridges in the solution space, then the heuristic may get stuck in a local minima. In practice, we expect servers to be relatively expensive items and our experiments show that the solution space is reasonably "flat". Hence, starting the ADD/DROP heuristic from the minimum number of servers works well.

The ADD/DROP greedy search can be summarised (following the model above) as follows:

Step 1 (Initialise): Randomly select the minimum allowed number of servers from the set of potential server locations. Assign client nodes to their nearest server. Design the network using the CLE procedure (see Chapter 4) and calculate the solution cost. Repeat for each possible combination of the minimum allowed number of servers from the set of potential server locations. Select the solution with the minimum cost as the current solution.

Step 2 (Search for next step): For each potential server location, if there is a server at that location drop (i.e. remove) it, otherwise add a server to that location. Assign client nodes to their nearest server. Design the network and calculate the solution cost. Select the solution with the minimum cost as the current solution.

Step 3 (Greedy improvement): As for Step 3 in Section 5.3.

5.3.2 ADD-*k* Heuristic

An alternative to the ADD/DROP heuristic is an ADD-*k* heuristic adapted from [Gavi92a]. The ADD-*k* heuristic generates a set of potential next step solutions by adding all combinations of 1 to k_{max} servers to the current solution (k_{max} is an input

to the algorithm). If the number of free (i.e. no server located at it) potential server locations is less than k_{max}, the search is limited to adding only as many servers as the are free locations. Each iteration finds the best k servers to add to the current solution (where $1 \leq k \leq k_{max}$). Servers are placed at these locations and these locations are removed from the set of free potential server locations. The search finishes when either there are no more free potential server locations, or no improvement on the current solution can be found.

Unlike the ADD/DROP heuristic, the ADD-k heuristic must start from the minimum number of servers allowed. This is because it only adds servers to the current solution, it does not remove them and hence has no way to search for solutions with fewer servers than the number it starts with.

The greedy search using the ADD-k heuristic starts in the same way as the ADD/DROP algorithm, by generating all possible combinations of the minimum allowed number of servers.

The ADD-k greedy search can be summarised (again, following the model above) as follows:

Step 1 (Initialise): As for Step 1 of ADD/DROP greedy search (Section 5.3.1).

Step 2.1 (Initialise): Let k equal 1, C be the cost of the current solution, P be the set of potential server locations, A be the set of potential server locations with servers located at them, and let U .be $P - A$.

Step 2.2 (Find the best k servers to add to the current solution):
Let $bestC_k \leftarrow C_k \leftarrow C$.
For $i \leftarrow k$ to $\min(k_{max}, |U|)$ do
    For each combination of i from U servers, add those servers
    to the network. Assign client nodes to their nearest server,
    design the network and calculate its cost, C_k. If $C_k < bestC_k$,
    let S_k be the set of servers added, and $bestC_k \leftarrow C_k$.
end loop.

Step 3 (Greedy improvement: Add the best k servers to the current solution):
If $bestC_k < C$,

$A \leftarrow A \cup S_k$,

$U \leftarrow U - S_k$,

If $i = k_{max}$, then $k \leftarrow 1$ else $k \leftarrow k_{max} - i$,

If $U \neq \varnothing$, then go to Step 2.2.

end if.

Finished, best solution has cost C.

The last statement in Step 3 before the algorithm checks if there are more potential server locations unused and loops back to Step 2.2, (If $i = k_{max}$ then $k \leftarrow 1$ else $k \leftarrow k_{max} - i$) ensures that the search in the next iteration does not overlap the previous one if less than k_{max} servers were added in the previous iteration.

5.4 Complexity and Execution Time Analysis

Figure 5.1 compares the execution time of the design procedures as the network size increases (the results for the design of networks with existing link topologies are similar). Two versions of the clustering procedure are included. The slowest one, which, of the variations of the clustering procedures, produces the best designs, and the fastest, which produces slightly more expensive designs. As our results in Section 6.4.1 show, the difference in the cost of the solutions produced by the fastest and slowest clustering algorithms (variations **A** and **G** in Table 5.1 respectively) is usually less 1%. However, the difference in execution time of the two variations is significant.

We see that the execution time of the ADD/DROP procedure is approximately an order of magnitude slower than the faster clustering procedure for larger networks. The ADD-k ($k_{max} = 2$) procedure is an order of magnitude slower again to design a network with 20 potential server locations (the difference increases as k_{max} is increased). In addition, the difference in execution time between the procedures increases with the number of potential server locations, or network size. The results for the ADD-k procedure are truncated due to limitations on the CPU time available. Despite this, the trends in the relative execution times of the procedures are clear from Figure 5.1. Our experiments have shown that solving a network with 25 poten-

tial server locations using the ADD-k procedure takes in the order of 24 hours, thus making its use impractical for networks any larger.

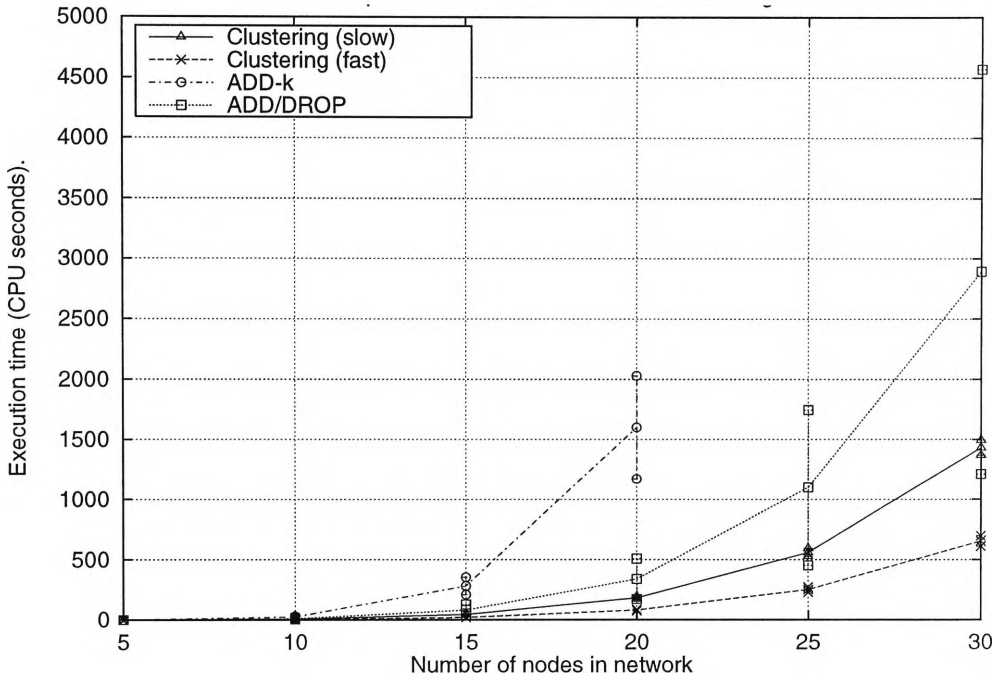


Figure 5.1 Execution time comparison for randomly generated networks without preexisting link topologies.

The error bars in Figure 5.1 represent the 95% confidence interval generated by recording the execution time of each procedure when applied to the same 30 randomly generated design problems of each size. The size of the design problems vary from 5 to 30 nodes, in 5 node increments. All nodes are considered as potential server locations.

The clustering algorithm is O(n) as it forms 1 to n clusters, where n is the number of potential server locations in the network. The best design is selected from that set.

In each step of its search the ADD/DROP heuristic evaluates

$$S = \binom{n-k}{1} + \binom{k}{1} = n \tag{5.3}$$

solutions, where n is the number of potential server locations and k is the number of servers in the current solution. In each step of its search the ADD-k heuristic evaluates

$$S = \binom{n-k}{1} + \ldots + \binom{n-k}{\min\{k_{\max}, n-k\}} \tag{5.4}$$

solutions. Say there are x states between the current solution and the optimum (and no ridges, so both heuristics will find it). The ADD/DROP heuristic will take x steps to find the minimum, evaluating xn solutions in the process. In the best case the ADD-k procedure will take $x/(\min\{k_{\max}, n-k\})$ steps, evaluating

$$\frac{x}{\min\{k_{\max}, n-k\}}\left[\binom{n-k}{1} + \ldots + \binom{n-k}{\min\{k_{\max}, n-k\}}\right] \tag{5.5}$$
$$= \begin{cases} xn, & k_{\max} = 1 \\ 2^n, & k_{\max} \geq n-k \end{cases}$$

in the process. Thus, the ADD/DROP heuristic is an order O(n) algorithm, while the ADD-k heuristic is between O(n) and O(2^n) as k_{max} varies from 1 to n (where n is the number of potential servers locations in the network).

It is worth noting that the execution time of the clustering procedures can be significantly reduced (e.g. halved), if the maximum possible number of servers is limited to $n/2$ for example. In most of our experiments we assume that servers cost approximately half as much as a 500 km link of equivalent capacity (500 km is the approximate average link length in randomly generated networks spanning a 1,000 km^2 area). In this context, our experiments show that the best solution usually contains less than $n/2$ servers, so limiting the number of servers to at most $n/2$ will not affect the quality of the solutions found. If the relative cost of components merits it, this limit could be altered. That is, if servers are expensive relative to links the limit should be reduced, if they are much less expensive than links it should be increased. This limit is employed in [Diri77] to improve the execution time of their clustering procedure.

It is important to note that the execution time of all of the design procedures is most dependant on the number of potential server locations rather than the total number

of nodes in the network. For simplicity it is assumed that all nodes in a network are potential server locations in most of our experiments. The addition of nodes that are not potential server locations to the network adds to the size of the link topology design problem, not the number of solutions examined by each procedure. Thus the total number of population centres (i.e. nodes) that a network spans may be significantly larger than the number potential server locations.

5.5 Heuristic Quality Assurance

There are two approaches that may be taken to obtaining a lower bound on the cost of the optimal solution to the primal problem formulated in Chapter 3. The first approach is to solve the primal problem directly. However, the primal problem is an extension of the general network design problem which has been shown to be NP-hard [Gare79]. This means that the only way we can be certain of finding the optimal solution is via an exhaustive search of the solution space. This is impractical for any but the smallest of problems $(n \leq 10)$ as it requires the examination of $2^n - 1$ possible solutions. The second approach is to formulate and solve a less complex but related problem whose optimal solution cost we know to be a lower bound on the cost of the optimal solution to the original problem.

5.5.1 Formulation of a Lower Bounding Problem

We formulate a less complex lower bounding problem by relaxing the original problem in two ways. The first set of relaxations transforms the non-linear constraints in the original problem into linear constraints. The second relaxation reduces the problem by ignoring the effect of inter-server traffic on the network. We observed in Chapter 3, that the DSNDP can be viewed as two interacting multicommodity flow problems. A reduced problem can be defined in which we ignore the effect on the network and the cost of transporting the inter-server traffic (its effect on servers is maintained). The cost of the optimal solution to this reduced server location and access network design problem will be a lower bound on the cost of a solution which includes inter-server traffic as well.

These two relaxations transform the original combinatorial optimisation problem into a less complex concave NLP optimisation problem whose solution is a lower bound on the original problem.

We also relax the stepwise nature of the objective function with respect to link and server capacities, thus allowing links and servers to take up any capacity.

As is common in the design of communication networks constraint (3.10) uses the average packet queueing delay, T, to ensure acceptable performance is obtained from the network [Klei76] [Gerl77] [Dutt92]. Ideally, we want to constrain T to be less than some value T_{max}.

As in Section 4.3 we use Dutta's technique to reformulate constraint (3.10), which is a non-linear function of the capacity and flow variables, into a constraint on link utilisation:

$$\frac{F_{ij}}{c_{ij} - F_{ij}} \leq \frac{\gamma T_{max}}{|L|}, \forall (i,j) \in L \tag{5.6}$$

where

$$\gamma = (1 + \delta) \sum_{i \in N} \gamma_i \tag{5.7}$$

$$\Psi F_{ij} \leq c_{ij}, \forall (i,j) \in L \tag{5.8}$$

$$\Psi = \frac{|L'|}{\gamma T_{max}} + 1 \tag{5.9}$$

Thus, constraints (3.10) and (3.11) can be replaced by (5.8). Similarly, allowing continuous server capacities means that constraint (3.13) can be replaced by:

$$x_k(1 + \delta) \leq w_k, \forall k \in P \quad . \tag{5.10}$$

Removing the inter-server traffic from the network allows us to drop constraints (3.9) and (3.16), and means that the total traffic flow on a link F_{ij}, is given by:

$$F_{ij} = f_{ij} + f_{ji}, \forall (i,j) \in L \tag{5.11}$$

Having linearised and reduced original problem, the lower bounding problem can be formulated (using the notation defined in Chapter 3) as follows:

$$\min\ Z(c,w) = \sum_{\{i,j\}\in L}\left\{\beta_1 c_{ij}^{\alpha_1} + \beta_2 c_{ij}^{\alpha_2} + \left(\beta_3 c_{ij}^{\alpha_3} + \beta_4 c_{ij}^{\alpha_4}\right) l_{ij}\right\} + \sum_{k\in P}\left\{\beta_5 w_k^{\alpha_5} + \beta_6 w_k^{\alpha_6}\right\} \tag{5.12}$$

Subject to:

$$\gamma_j + \sum_{\{j\ni(i,j)\in L\}} f_{ij} - \sum_{\{j\ni(j,i)\in L\}} f_{ji} - x_j = 0,\ \forall i\in N,\ (x_j = 0,\ \forall j\notin P) \tag{5.13}$$

$$\Psi(f_{ij}+f_{ji}) \le c_{ij},\ \forall (i,j)\in L \tag{5.14}$$

$$c_{ij} \le C_{ij},\ \forall (i,j)\in L \tag{5.15}$$

$$x_k(1+\delta) \le w_k,\ \forall k\in P\ . \tag{5.16}$$

$$w_k \le C_k,\ \forall k\in P \tag{5.17}$$

$$f_{ij} \ge 0,\ \forall (i,j)\in L \tag{5.18}$$

$$x_k \ge 0,\ \forall k\in P \tag{5.19}$$

$$w_k \ge 0,\ \forall k\in P \tag{5.20}$$

$$c_{ij} \ge 0,\ \forall (i,j)\in L \tag{5.21}$$

5.5.2 Solving the Reduced Lower Bounding Problem via Continuous Branch & Bound

The server location and access network design problem, formulated above, is a single commodity flow version of the concave multicommodity flow lower bounding problem defined in Section 4.3. We solve this problem using the same continuous branch-and-bound method described in Section 4.4.

One feature of the continuous branch-and-bound algorithm from [Ryoo95], not discussed in Section 4.4, is a tolerance factor on the aggressiveness of the decision to bound a branch of the search tree. As observed previously, it is known that branch-and-bound algorithms often find optimal (or near optimal) solutions very quickly, but take a long time to verify the fact by searching the rest of the search tree. When-

ever the underestimating problem associated with a node in the search tree is solved, the lower bound in that portion of the solution space, *LB*, is compared with the best known upper bound of the problem, *UB*. If $LB \geq UB - \varepsilon$ the node is considered to represent an uninteresting portion of the search tree and discarded (a similar process occurs when a new global best upper bound is found). In [Ryoo95] ε is defined as being greater than zero and such that *LB* and $UB - \varepsilon$ can be considered "close." The larger the value of ε, the more likely the search tree will be bound at a given node.

The effect is that in the best case the search finds the optimal solution quickly and will simply discard solutions that are close to it. In the worst case, the solution produced by the branch-and-bound procedure will be within at most ε of the true global minima. In an environment in which the solution space is relatively flat near the global minima, the former is more likely to occur.

A disadvantage of the tolerance factor is that it has a significant effect on the execution time of the algorithm only towards the end of its search. This means that it does not affect the execution time of large problems such as the Concave TCFA lower bounding problem considered in Chapter 4. However, the tolerance factor is useful when applying the algorithm to solving the lower bounding problem for the DSNDP, because the solution space of the DSN lower bounding problem is considerably smaller.

Rather than employing an absolute value ε, we let ε equal some fraction of the current *UB*. The question remains what value of ε should be used to gain the maximum benefit, without unduly affecting the quality of the solutions produced.

Figure 5.2 shows how the memory (left-hand vertical axis) and execution time (in CPU seconds on the right-hand vertical axis) requirements of the continuous branch-and-bound procedure are affected by the value chosen for ε. The amount of memory consumed by the branch-and-bound procedure is determined by the maximum size of the list of unexplored nodes in the search tree (labelled as the B&B list in Figure 5.2). The estimates shown are the means taken from solving 30 randomly generated 10 node DSNDP's with pre-existing link topologies. The results for

experiments without pre-existing link topologies are similar. The error bars represent the 90% confidence interval for each estimate.

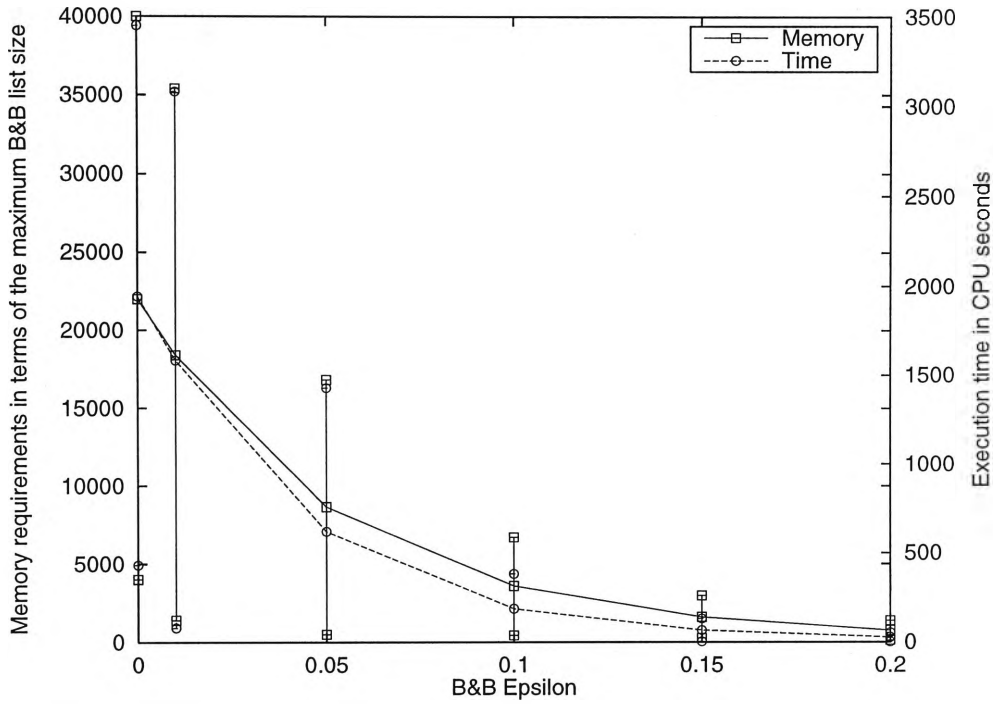


Figure 5.2 Memory and execution time requirements of the continuous branch-and-bound procedure used to solve the DSN lower bounding problem vs. the bounding aggressiveness parameter, ε.

Figure 5.2 shows that both the memory requirements and execution time of the procedure can be reduced by an order of magnitude or more, if a value of $\varepsilon \geq 0.1$ is employed.

Figure 5.3 again shows how the continuous branch-and-bound procedure execution time varies with ε. In addition, Figure 5.3 shows how the cost of the solutions produced by the procedure are affected by the value of ε. The cost of the solution produced by the procedure using each value of $\varepsilon > 0$ has been normalised with respect to the cost of the solution produced when $\varepsilon = 0$. Again, the estimates shown are the means taken from solving 30 randomly generated 10 node DSNDP's with pre-existing link topologies. The results for experiments without pre-existing link topologies are also similar. The error bars represent the 90% confidence interval for each estimate.

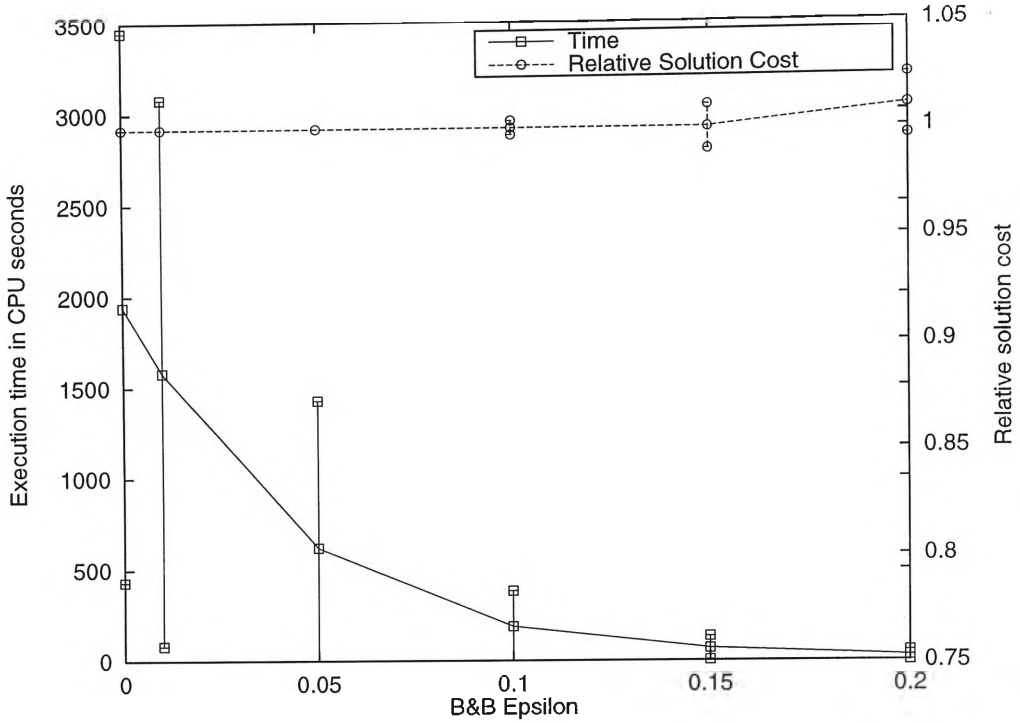


Figure 5.3 Execution time requirements of, and relative cost of solutions produced by, the continuous branch-and-bound procedure used to solve the DSN lower bounding problem vs. a range of values of the bounding aggressiveness parameter, ε.

The results in Figures 5.2 and 5.3 indicate that setting $\varepsilon \leq 0.15$ has little (less than 1%), or no, impact on the cost of the solutions found, but reduces the memory and execution time requirements of the procedure by up to two orders of magnitude. These results lead us to employ a value of $\varepsilon = 0.15$ in the remainder of our experiments that use the continuous branch-and-bound algorithm to determine a lower bound on the cost of an optimal DSN design. Our experiments on the effect of altering the value of ε with networks of varying sizes showed similar results as those presented above.

5.6 Conclusions

In this chapter we have described three heuristic procedures for solving the DSNDP. The clustering procedure employs the most knowledge of the problem domain, while the ADD/DROP and ADD-*k* procedures employ very little knowledge of the problem domain. The ADD-*k* procedure attempts to overcome potential problems associated with an ill behaved solution space. However, our examination of the

computational complexity of the procedures shows that the ADD-*k* procedure suffers a significant complexity and execution time penalty to do so, thus making it impractical for solving problems with more than 25 potential server locations.

In particular, our examination of the execution time of the three procedures confirmed that the ADD-*k* procedure takes an order of magnitude longer to design a network with 20 potential server locations than the ADD/DROP procedure, which in turn takes an order of magnitude longer than the fastest clustering procedure.

We have also described how a DSNDP can be relaxed and reformulated to define a less complex problem whose solution will be a lower bound on the cost of the optimum solution to the original problem. In addition, by trial and error we have found a value for the branch-and-bound procedure tolerance factor, ε, which reduces the execution time by an order of magnitude without significantly affecting solution quality.

Having described three algorithms for solving the DSNDP, it remains to compare the cost of the solutions they produce. An extensive performance comparison of the three DSN design procedures is the subject of the next chapter. In addition, we use the description of our experiments to outline a methodology that a designer may follow to use our procedures to design a real DSN.

6. Design Methodology & Performance Comparison

Wisdom denotes the pursuing of the best ends by the best means.

- Francis Hutcheson.

6.1 Introduction

In the previous chapter we described three procedures for designing DSN's. The complexity and average execution time of the three procedures were also compared. This chapter compares the cost of the solutions produced by the three DSN design procedures. In addition, we describe a DSN design methodology using our procedures.

6.2 A Distributed Server Network Design Methodology

There are three major types of inputs required before the design of a DSN can be undertaken using the procedures described in the previous chapter.

The first of these describes the types of network components available and their cost. In the context of a DSN, there are two types of network component - servers and links. Both servers and links come in a range of capacities, which needs to be reflected in their cost functions.

The second design input required is a description of the traffic requirements in the network. This information is required as an input to the CLE procedure, which designs the link topology of the DSN. In a DSN, the traffic requirements take two forms. The first set of requirements are derived from static aspects of the problem.

That is, the location of client nodes, and the volume of client-server traffic each will generate. The second set of requirements are dynamic and depend on the number of servers, their location, the assignment of client nodes to servers, and the level of interaction between servers. Once all these factors are known, a complete description of the traffic requirements can be generated and the link topology designed.

The third class of design inputs serve to constrain the designs which are produced. Constraints may be placed on the performance, or reliability of the designs produced. Logistic and/or political constraints may also be included, for example, constraints on the potential sites for servers, the links (not) allowed between nodes, the maximum (minimum) number of servers to be employed.

The specific design inputs employed in our experiments are described in the following subsections. However, it should be stressed that the DSN model, and our design procedures are flexible. A wide variety of applications can be catered for by employing different, cost functions, methods to generate the traffic requirements in the network, and/or design constraints. Some of these variations of the DSN model and how our design procedures might be enhanced to allow for them are discussed in Section 6.2.4.

6.2.1 Component Cost Function Parameters

In this dissertation we have assumed that network component costs are concave with respect to capacity to reflect the effects of economies of scale. The exact form of the double power-law cost functions is given in Chapter 3 (see Equations (3.3), (3.4) and (3.5)).

Table 6.1 shows the costs assumed in our experiments for typical server capacities. See Chapter 4, Table 4.1 for the link costs assumed.

Table 6.1 Server Capacities and Corresponding Costs

Capacity (Mbps)	Cost ($/year)
300	25,000
600	35,000
900	42,000

For the purposes of assigning server capacities in the network design we assume that server capacity is available in 300 Mb/s increments.

Table 6.2 shows the resulting parameter values when the double power-law cost functions (Equations (3.3), (3.4) and (3.5)) are fitted to the component cost data from Table 6.1. See Chapter 4, Table 4.2 for the equivalent link costs parameters.

Table 6.2 Double power-law cost function (Equations (3.3), (3.4) and (3.5)) parameters fitted to cost data.

	β_5	α_5	β_6	α_6
Server	550	0.6	10,000	0.06

The effect of varying the relative cost of links and servers on the performance of the design algorithms, is examined in Section 6.4.2.2.

6.2.2 Client Demand Topology and Volume

In order to perform a comprehensive comparison of the various design procedures presented in this dissertation, most experiments use randomly generated "typical" network design problems. The results from applying the DSN design procedures to a specific network, taken from [Saha95], are included in Section 6.3 to illustrate the typical output of the procedures.

As before, we generate random network design problems by randomly distributing nodes. Each node has a population sampled from a Pareto distribution, as outlined in Section 4.7.1.

As discussed in Chapter 3, to capture the effects of inter-server communication on the network, we assume that some percentage, δ, of the traffic received by any given server is distributed uniformly amongst the other servers in the network (see Equation (3.1)). For the purposes of most of our experiments we assume $\delta = 0.1$. That is, 10% of the traffic offered to each server by its clients is copied to the other servers in the network. The effect of varying δ, on the performance of the design algorithms, is examined in Section 6.4.2.3.

6.2.3 Design Constraints

The mathematical formulation of the DSNDP in Chapter 3 included a constraint on the maximum average queuing delay incurred by a message, T_{max}. In Chapter 4 we described how that constraint can be transformed into an equivalent constraint on the maximum link utilisation. The CLE procedure is able to enforce either of these types of constraints. Indeed, the CLE procedure is flexible enough that additional (or different) network performance constraints could easily be introduced. However, for our experiments we have chosen to enforce the constraints on the average maximum queuing delay only. That is, we require that $T_{max} \leq 100$ milliseconds. Altering this value affects all design procedures equally, hence we have used this single T_{max} value only.

As discussed in Chapter 1, we consider two scenarios with respect to the link topology of the final design. In the first case we assume that there are no final link topology restrictions. Given that any link is allowed in the design, the initial topology is a full mesh over the N nodes. In the second case, we assume that the final link topology is restricted to a subset of an initial given topology. This may be due to a need to employ an existing link topology, or logistic/political constraints that make links between certain pairs of nodes infeasible. In our experiments, existing link topologies are generated by designing a backbone link topology connecting all nodes using the CLE procedure described in Section 4.7.1.

In most of our experiments we have assumed that all client nodes are also potential server locations, and that there are no constraints on the minimum, or maximum, number of servers allowed. The effect of limiting the possible server locations in a network on the performance of the design algorithms, is examined in Section 6.4.2.4.

6.2.4 Further DSN Variations

As discussed above while the DSN model that we have proposed is flexible and can be applied to a wide variety of services we have for practical purposes limited our investigations to certain types of DSN. In the this section we discuss some varia-

tions of the DSN model that might appear in practice and how our design procedures could be enhanced to allow for them.

6.2.4.1 DSN's with Non-Uniform Inter-Server to Client-Server Traffic Ratios

We have assumed that proportion of client-server traffic copied onto the inter-server network by each server (to each other server) is uniform across all servers. The impact to the DSN design procedures is relatively minor if this is not the case. The proportion of client-server traffic copied onto the inter-server network by each server is used as an input to the design process once a set of server locations has been decided upon and clients have been assigned to servers. It is used to generate an O-D traffic requirements matrix that specifies the traffic flow between all pairs of nodes. If the traffic proportions varied then the O-D traffic matrix is altered and the subsequent Concave TCFA problem has a slightly different input. None of the design procedures presented in the dissertation would be adversely affected by the introduction of variable proportions client-server to inter-server traffic per server.

6.2.4.2 DSN's and Background Peer-to-Peer Traffic

We have assumed the design of a DSN is unaffected by the requirements of other services within the underlying network due to the assumed use of Virtual Service Networks (VSN's). If VSN's were not available, the DSN traffic would have to co-exist with, for example, background peer-to-peer traffic flowing between the nodes in the network. The existence of background traffic in the network would mean that the cost of transporting a given volume of traffic over a link cost would depend not only on the capacity requested and length of the link, but also on the volume of traffic already present on the link.

The inclusion of background traffic in the DSNDP would have implications for the use of the link distance between nodes in the "pull" function of the clustering procedure. The use of link distance assumes that all links have the same cost function parameters. Hence, the cost of transporting a given volume of traffic over different of links varies in proportion to the length of the links only. If the link cost per unit capacity varies across links, then the procedure could use the cost of transporting an

arbitrary volume of traffic between the nodes of interest, instead of the link distance between them.

In contrast, neither of the greedy search based DSN design procedures, the ADD/DROP and ADD-*k* procedures, would be affected by the inclusion of background traffic in the network as they do not make use of any link cost or inter-node "distance" information when searching for new arrangements of servers. They simply rely on the results of the Concave TCFA problem solved in each iteration to evaluate the quality of each solution found.

For all three design procedures the background traffic would simply be included in the O-D traffic requirements matrix given as an input to each Concave TCFA problem solved.

6.2.4.3 DSN's with Multiple Client to Server Allocation

We have assumed that clients are assigned to only one server, and that any interaction with another server takes place via that server. An interesting addition to the DSNDP would be the requirement that clients be assigned to, or be able to connect to, multiple servers either for reliability or performance. The implications to the design process depend on the reason for the multiple allocation of clients to servers.

If the intention is to enhance the reliability of the DSN, we may assume that under normal operating conditions, clients would communicate with just one server. The connection to a second server would be used only in the event of a link or server failure. Then the existing design procedures already allow for this. By including a topology constraint in the CLE network topology design procedure such that any node is guaranteed to have at least two diverse paths to at least two different servers the reliability requirement could be satisfied.

However, if the intention is to allow each client to divide its communication across multiple servers to enhance performance, then the server location-allocation portions of each of the design procedures must be modified. In the ADD/DROP and ADD-*k* procedures client nodes could simply be assigned to the N nearest servers, where N is the number of servers with which each client is to communicate directly. The clustering procedure could easily be altered to select N servers within each clus-

ter. In all three procedures diverse paths between clients and their servers may also be guaranteed by the CLE procedure for reliability.

6.2.4.4 DSN's to Support Multiple Services

We have assumed that all servers are equal, and as such clients can be allocated to any potential server location. This would no longer be the case if we were designing a DSN to support multiple services. Potentially overlapping subsets of clients would then need to be assigned to the different types of servers. Each type of server would have its own set of potentially overlapping possible server locations.

The implication for the design procedures of including multiple services into a single DSN is effectively limited to the server location-allocation phase of the design process. In the ADD/DROP and ADD-*k* procedures the addition, and deletion in the ADD/DROP procedure, of servers would have to iterate over the possible server locations for all services, rather than just the one set of potential server locations. The effectiveness of the procedures should not be affected, simply their execution time. The implications on the usefulness of the procedures would have to be determined through further research.

The clustering procedure could be used in either of two ways to design a DSN to support multiple services. In the simplest, case each service could be treated separately whilst clustering nodes and selecting server locations, with the traffic requirements for all services then combined to design the supporting network topology. However, a more effective method would be to alter the clustering procedure such that it is able to take advantage of economies of scale by creating intersecting clusters and possibly co-locating servers. The development of such a clustering procedure and the performance comparison with the simpler approach discussed would be an interesting area for further research.

6.2.4.5 Hierarchical DSN's

Thus far we have assumed that all servers are equal. Clients communicate with any server and any server can communicate with any other thus making the inter-server network a flat peer-to-peer network. We have not considered the case where the DSN consists of multiple server types. This would produce an environment where

clients communicate with one class of server, servers of a given class communicating amongst themselves, and also with other classes of server and so on, thus creating a hierarchical inter-server network.

As above, the ADD/DROP and ADD-*k* procedures could be extended to iterate over *N* types of server addition (or deletion), where *N* is the number of possible different classes of server or tier in the hierarchy. The execution time implications of this approach and the details of how the procedures might determine the most appropriate number of tiers in the server hierarchy remain to be investigated.

There is already some work in the literature, e.g. [Diri77], on hierarchical hub location problems which may be useful in enhancing our clustering based procedure to cope with a hierarchical DSN. The simplest approach would be to apply the clustering algorithm iteratively. This would cluster nodes in the lowest layer of the hierarchy then treat each cluster as a single node in the next tier up and cluster them accordingly. The procedure would have to be iterated to determine the best number of clusters to form at each layer of the hierarchy. Furthermore, if we assume that the highest tier consists of a single server (or a specified number of servers) the procedure would automatically determine the best number of tiers in the hierarchy. This is because the procedure would continue iterating until it determined that optimal number of clusters (hence servers) in a level was one (or the number given). Again the details of the procedure, its execution time and usefulness are areas for further research.

6.3 A Specific Network Example

Before investigating the average solution quality produced by our design procedures we illustrate the type of designs produced using a specific network example based on one that appears in the literature.

Figure 6.1 shows a 22 node network example from [Saha95], with a link topology generated using our CLE algorithm ([Saha95] does not provide a link topology as the authors consider geographic rather than link distances). The size of each node is proportional to its population, sampled from a Pareto distribution. Nodes that are potential server locations are shown by a double ring (all nodes in this example).

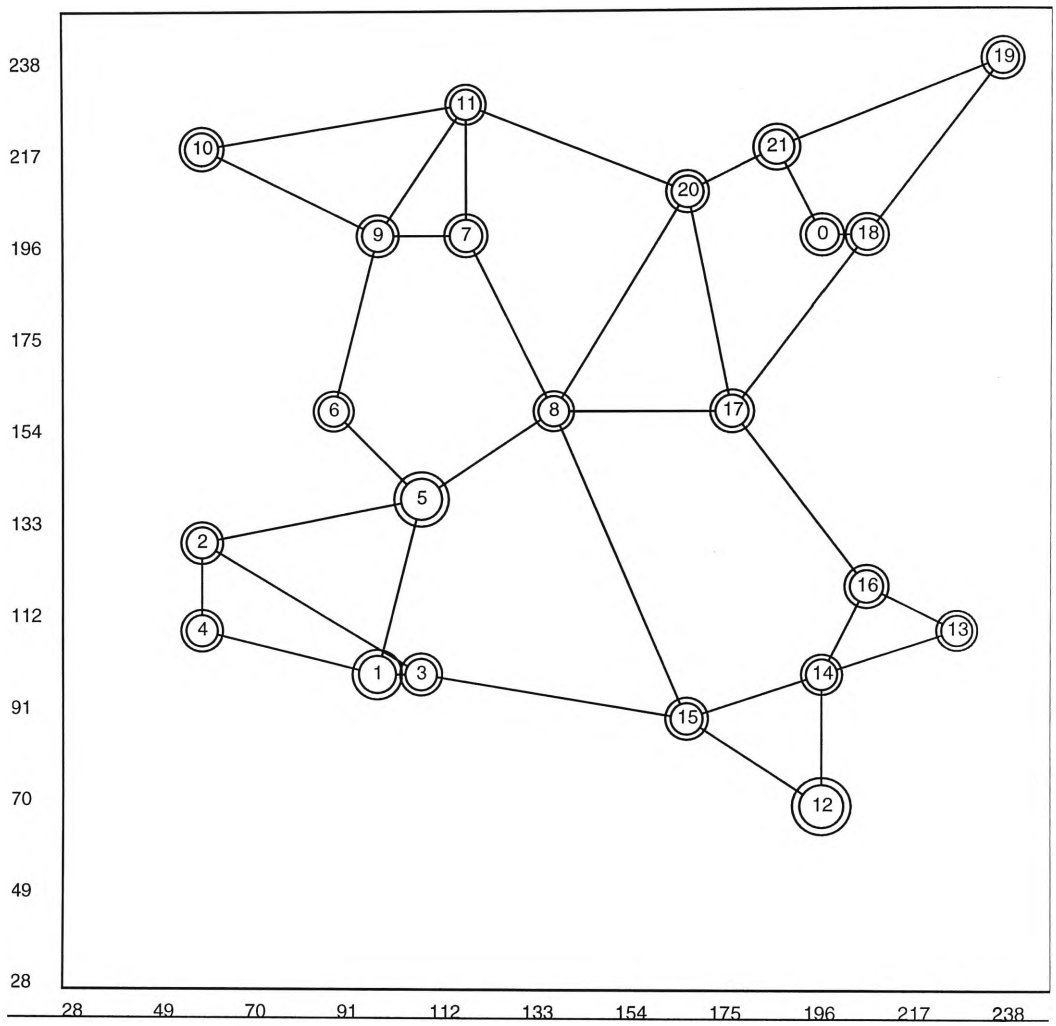


Figure 6.1 A specific 22 node network example from [Saha95], with a preexisting link topology generated by the CLE algorithm.

Tables 6.3 and 6.4 show the absolute and relative costs of the solutions produced by the DSN design procedures when applied to the 22 node network, with and without taking the existing link topology (Figure 6.1) into account. The results produced by the slow and fast variations of the clustering procedure (variations **A** and **G** in Table 5.1 respectively) are presented. For comparison, the results are normalised with respect to the cost of the solution produced by the slow clustering procedure. Also, included in these tables is the execution time of each procedure (on a 166 MHz Pentium™ PC), the number of servers in the final designs, and the number of potential solutions examined by each procedure.

To illustrate the type of designs produced by the procedures, Figure 6.2 shows the design of the 22 node network, with an existing link topology (Figure 6.1), produced by both slow and fast clustering DSN design procedures. The thickness of the

Table 6.3 22 node network, with an existing link topology, DSN design procedure results.

Procedure	Absolute Solution Cost ($)	Relative Solution Cost	Time (CPU seconds)	Number of Servers	Number of Solutions Examined
Clustering (slow - **A**)	1.02625×10^6	1.0	110	7	22
Clustering (fast - **G**)	1.02625×10^6	1.0	47	7	22
ADD/DROP	1.02297×10^6	0.9968	355	6	197
ADD-*k*	1.02297×10^6	0.9968	7211	6	3648

Table 6.4 22 node network, without an existing link topology, DSN design procedure results.

Procedure	Absolute Solution Cost ($)	Relative Solution Cost	Time (CPU seconds)	Number of Servers	Number of Solutions Examined
Clustering (slow - **A**)	1.03431×10^6	1.0	291	5	22
Clustering (fast - **G**)	1.03431×10^6	1.0	48	5	22
ADD/DROP	1.02834×10^6	0.9942	355	5	175
ADD-*k*	1.02834×10^6	0.9942	13690	5	3768

links shown is proportional to their capacity. Nodes selected as server locations are surrounded by a solid ring (nodes 1, 5, 7, 10, 12, and 21, in this example). Clusters of nodes (i.e. all nodes assigned to the same server) are indicated by the dotted lines.

Figure 6.3 shows the design of the 22 node network, with an existing link topology (Figure 6.1), produced by both the ADD/DROP and ADD-*k* DSN design procedures.

As our later results show, the fast clustering procedure (**G**) does not usually perform as well as the slower version (**A**). However, in this case the two clustering procedures produced the same results. Similarly, the ADD/DROP and ADD-*k* procedures also produce the same results both when the link topology is constrained and unconstrained. Again, our later results show that this is not usually the case, the ADD-*k* procedure usually produced slightly cheaper designs than the ADD/DROP procedure. It is interesting to note that the difference in the number of servers employed in the two solutions described in Table 6.3 (and illustrated in Figures 6.2 and 6.3), has very little (less than 1%) effect on the total cost of the solutions. This suggests that the DSNDP follows the trend for network design problems observed in the literature. That is, the solution space near the global minima in network design problems

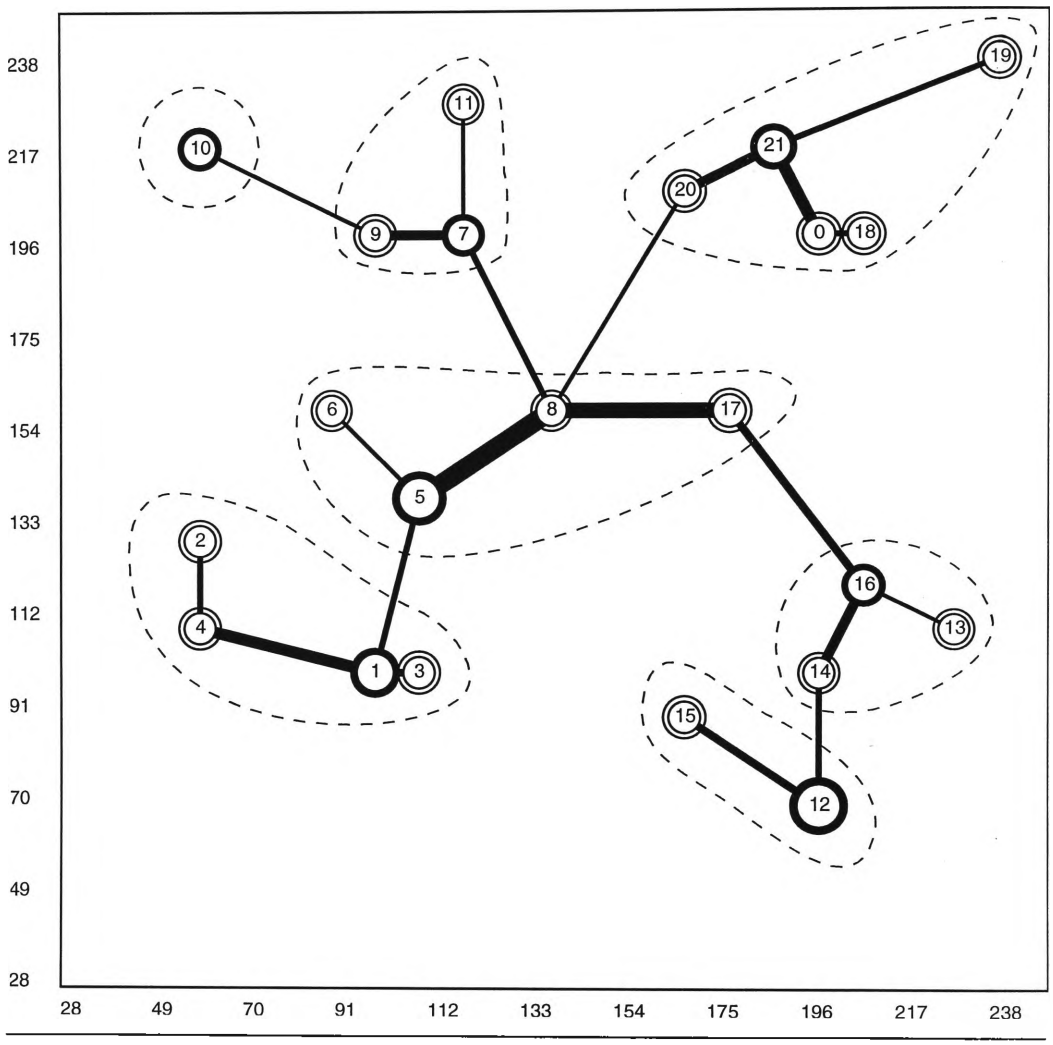


Figure 6.2 Design of 22 node network, with existing link topology, produced by both fast and slow clustering DSN design procedures.

is often relatively flat (see for example [OKe86] or [Butt96]). A key point to note is the significant difference in the execution time required by the procedures. This contrasts markedly with the relatively insignificant difference in the cost of the solutions they produced.

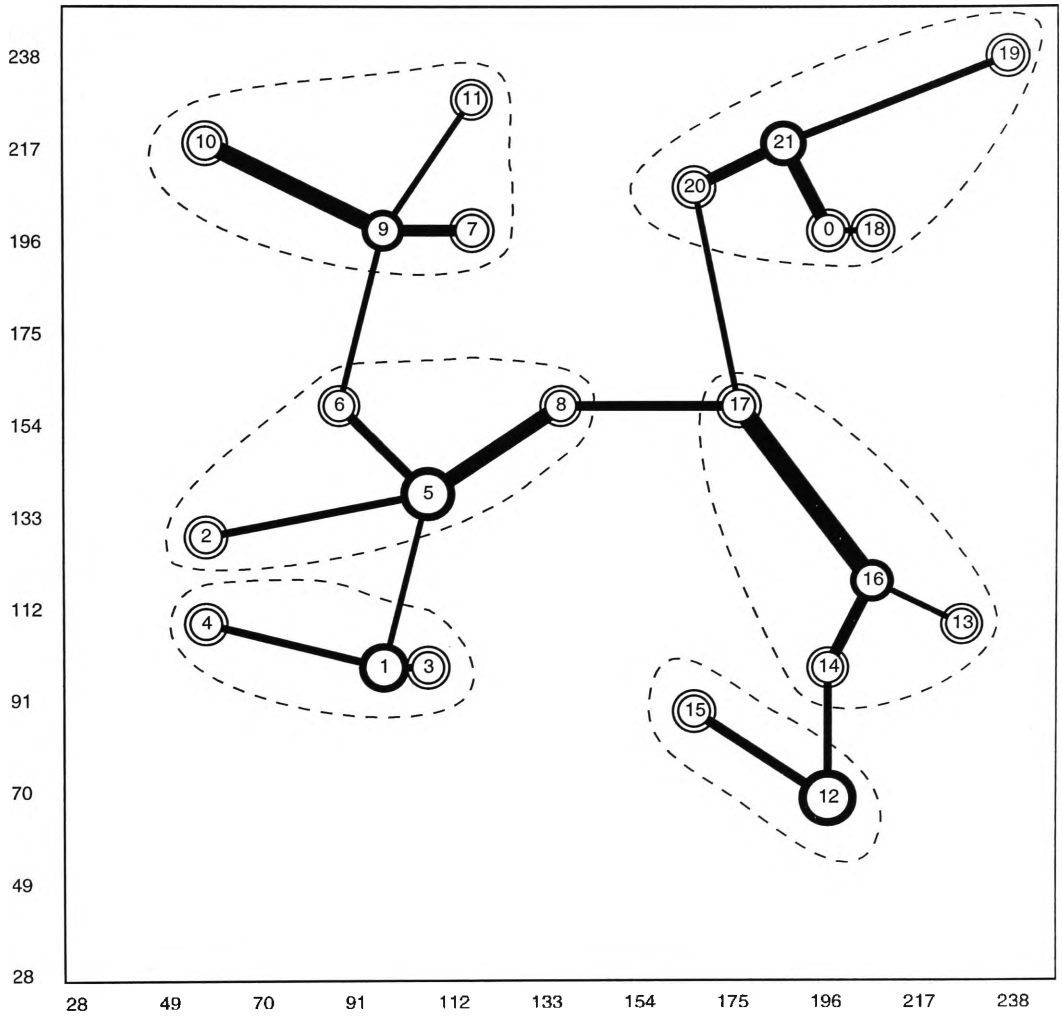


Figure 6.3 Design of 22 node network, with existing link topology, produced by both the ADD/DROP and ADD-*k* DSN design procedures.

6.4 Random Network Results

Having examined the performance of the DSN design procedures using a specific network example, we examine their average performance under a variety of conditions in this section. In Chapter 5 we described different ways to implement each phase of the clustering algorithm. Variations of the clustering procedure produced by selecting various combinations of the sub-procedures of the algorithm were described in Section 5.2.4. For clarity, we examined only the slowest and fastest variations of the clustering procedures (procedures **A** and **G** from Table 5.1 respectively) in our previous results. We compare the performance of all variations of the clustering algorithm in the following section (Section 6.4.1). Following this com-

parison, we return to considering the performance of only the slowest and fastest variations (**A** and **G**), when comparing the performance of the clustering procedures with the ADD/DROP and ADD-*k* DSN design procedures.

6.4.1 Performance Comparison of the Clustering Algorithm Variations

As described above, the node clustering based DSN design procedure consists of a number of different parts. For some parts of the procedure we discussed two alternative implementations. Here we examine the performance of various combinations of those implementation choices. The variations of the clustering algorithm were described in Section 5.2.4.

6.4.1.1 Network Size

Figure 6.4 compares the relative cost of designs produced by clustering procedures in Table 5.1, for a range of network sizes. To allow the comparison of the procedures, the cost of the solutions produced by each procedure have been normalised with respect to the cost of the solution produced by procedure **A**. The estimates in Figure 6.4 were obtained by designing 100 random networks of 5 through 20 nodes, and 30 of 25 and 30 nodes. All network nodes are considered as potential server locations.

In all experiments the error-bars represent the 95% confidence interval for each estimate. In addition, in the experiments in this section the link topologies of the networks were unconstrained. The corresponding results for all experiments in this section where the same networks were designed with constrained network topologies are given in Appendix A.

The results in Figure 6.4 show that for the most part there is very little difference in the cost of the solutions produced by the various clustering procedures.

Figure 6.5 compares the execution time (on a 166 MHz Pentium™ PC) of the variations of the clustering procedure (see Table 5.1) for the same range of network sizes as in Figure 6.4.

As can be seen in Figure 6.5, the most complex clustering procedures (procedures **A**, **B**, and **C**) take the most time. Thus if time is of the essence the

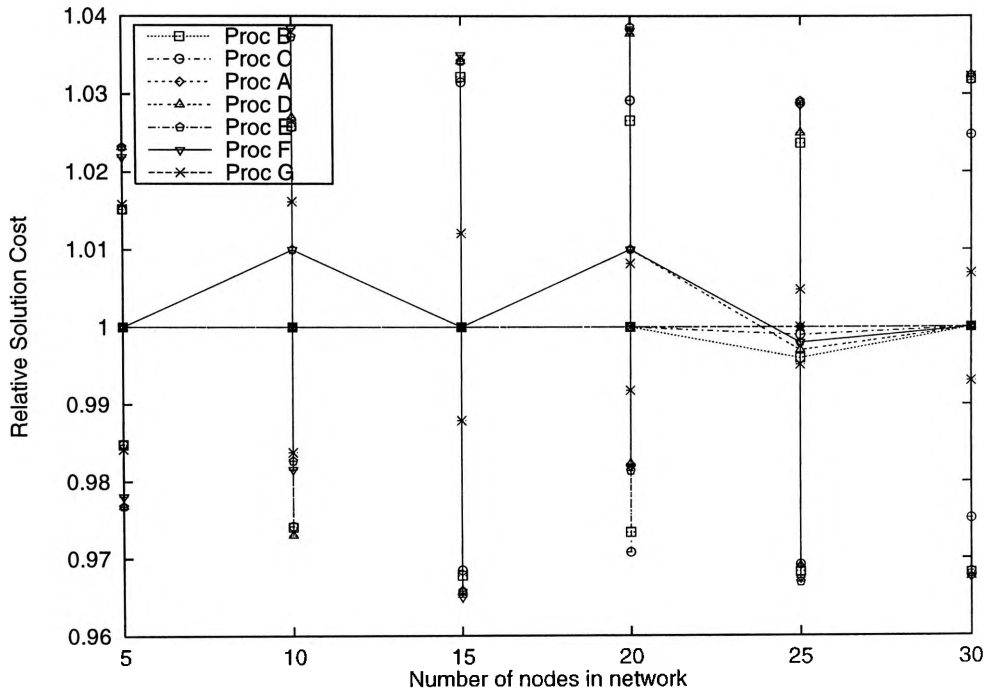


Figure 6.4 Relative cost of solutions produced by variations of the clustering procedure as the size of the network varies. Network topologies are unconstrained.

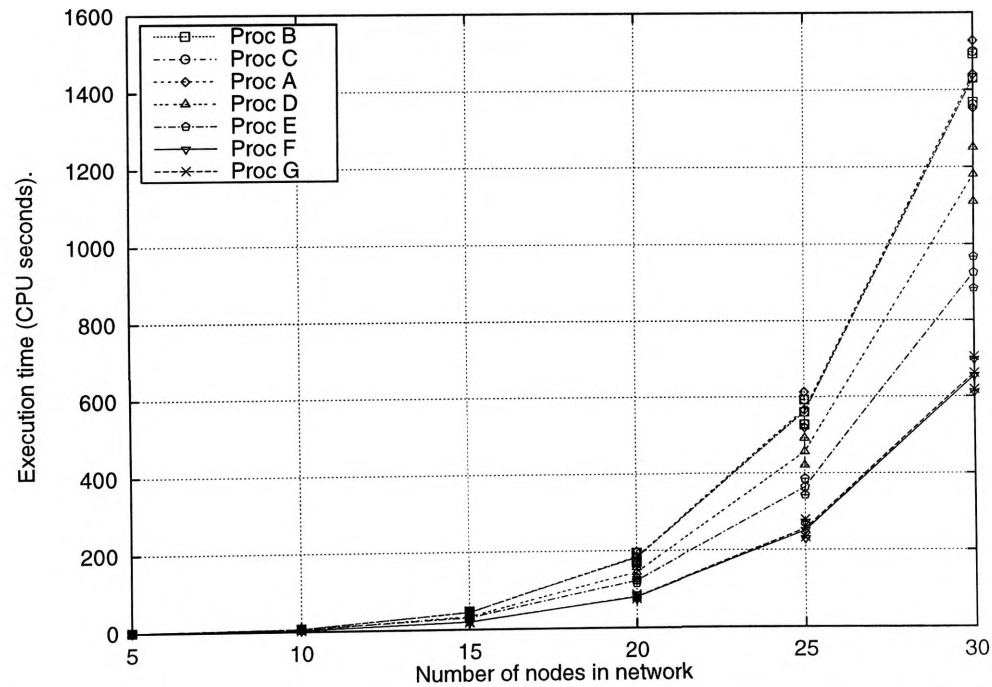


Figure 6.5 Execution time of variations of the clustering procedure as the size of the network varies. Network topologies are unconstrained.

designer would be unwise to use these procedures. Locating the servers at the CoM of each cluster (procedures **D**, **F**, and **G**), rather than searching for the best location (procedures **A**, **B**, **C**, and **E**), provides an improvement (approximately 14% to 22%) in execution time without significantly affecting the quality of the solutions produced. Designing the network link topology in parts (procedures **E**, **F**, and **G**) rather than as a whole (procedures **A**, **B**, **C**, and **D**) provides a larger (approximately 20% to 60%) improvement. Combining these two options, locating servers at the CoM and designing the network topology in parts (procedures **F** and **G**), provides the most improvement (approximately 60%) without significantly impacting solution cost.

6.4.1.2 Server to Link Cost Ratio

In previous results we have assumed a set of cost function parameters (Tables 4.2 and 6.2) that result in servers which are approximately half the price of an equivalent capacity, 500 km link (500 km is the approximate average link length in randomly generated networks spanning a 1,000 km^2). Figure 6.6 compares the relative cost of designs produced by variations of the clustering procedure (see Table 5.1) as the relative cost of servers and links varies in 10 node networks. Again, all nodes are considered potential server locations. The cost of the solutions produced by each procedure have been normalised with respect to the optimal solution generated via an exhaustive search of all possible server configurations (which is possible because of the small size of the problem (i.e. 10 nodes)). The estimates in Figure 6.6 were obtained by designing 100 random 10 node networks.

In Figure 6.6 we see that all of the variations of the clustering procedure produce designs whose mean cost is within 2% of optimal. As before, the faster algorithms (e.g. procedures **E**, **F**, and **G**) produce the highest cost solutions. However, as servers become more expensive relative to links, the designs tend to become more centralised, often employing only one server irrespective of which clustering procedure is used. Since all clustering procedures are good at finding solutions involving only one server the difference between them decreases as servers become more expensive.

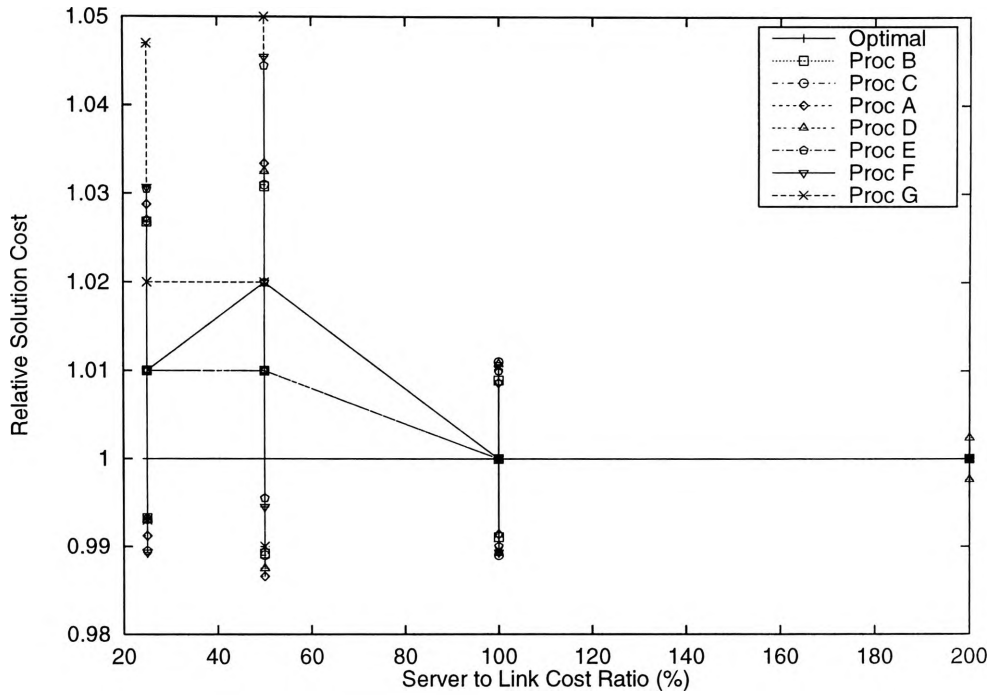


Figure 6.6 Relative cost of solutions produced by variations of the clustering procedure as the relative cost of servers and links varies. Network topologies are unconstrained.

6.4.1.3 Inter-server to Client-server Traffic Ratio

In our previous results we have assumed that 10% of the traffic delivered to each server by its clients is distributed amongst the others servers in the network (i.e. $\delta = 0.1$ in Equation (3.1)). Figure 6.7 compares the relative cost of designs produced by variations of the clustering procedure (see Table 5.1) as the ratio of inter-server to client-server traffic varies from 0% to 40% in 10 node networks. Again, the cost of the solutions produced by each procedure have been normalised with respect to the optimal solution generated by an exhaustive search. Futhermore all nodes are considered as potential server locations. The estimates in Figure 6.7 were obtained by designing 100 random 10 node networks.

The results in Figure 6.7 show that the performance of all of the clustering procedures are relatively insensitive to changes in the volume of inter-server versus client-server traffic. As usual the faster procedures occasionally do not perform as well (i.e. produce higher cost solutions) as their more complex counterparts. It is interesting to note that all of the procedures are able to consistently find the optimal solu-

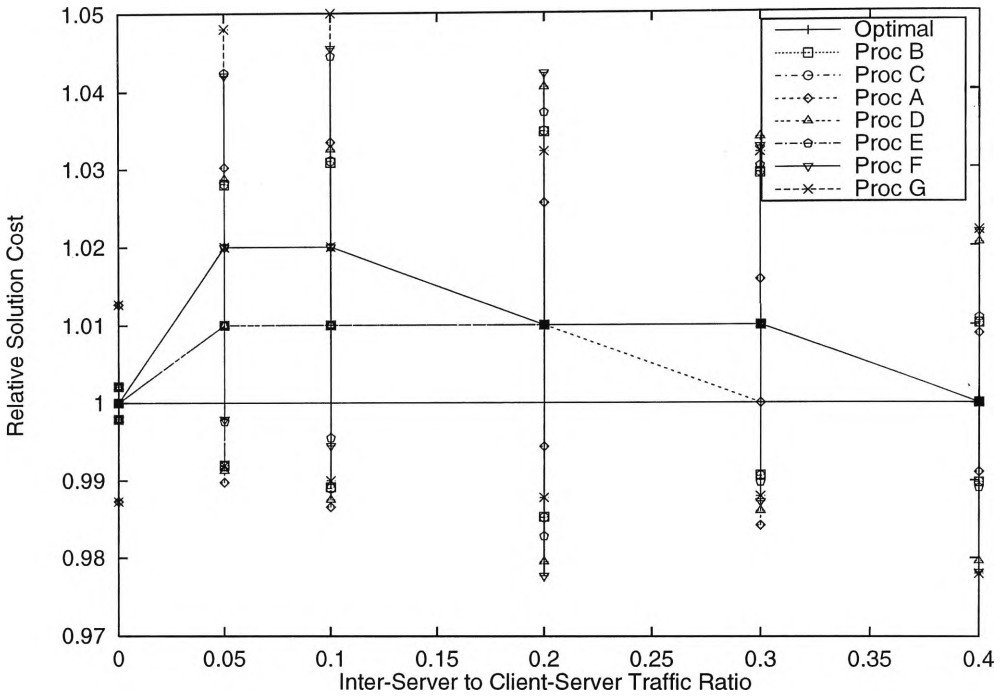


Figure 6.7 Relative cost of solutions produced by variations of the clustering procedure as the ratio of inter-server to client-server traffic varies. Network topologies are unconstrained.

tion when there is either no inter-server traffic (extreme left of Figure 6.7), or a large volume of inter-server traffic (extreme right of Figure 6.7). As observed previously, the power of the clustering procedure comes from its use of knowledge about the expected optimal solution to produce potential solutions. A key assumption built into the clustering procedures is that the optimal solution will contain clusters of nodes. In the absence of inter-server traffic there is nothing to perturb clustering of nodes, hence all of the clustering procedures perform well. When inter-server traffic dominates, the optimal solution becomes more centralised to minimise the amount of inter-server traffic. In the extreme only one server will be employed. As we observed in the previous section, the clustering algorithms are all good at finding single server solutions. Hence all of the clustering procedures perform well both when the volume of inter-server traffic is high and when it is low.

6.4.1.4 The Number of Potential Server Locations Relative to the Total Number of Nodes in the Network

In previous sections we have assumed that all nodes are potential server locations. Figure 6.8 compares the relative cost of designs produced by variations of the clustering procedure (see Table 5.1) as the number of potential server locations varies in

10 node networks. For each network generated we varied the number of potential server locations, P, from 1 to 10. Potential server locations were added to the network in decreasing order of population size (e.g. if $P = 3$ the three nodes with the largest populations were selected as potential server locations). Again, the cost of the solutions produced by each procedure have been normalised with respect to the optimal solution generated by an exhaustive search. The estimates in Figure 6.8 were obtained by designing 100 random 10 node networks.

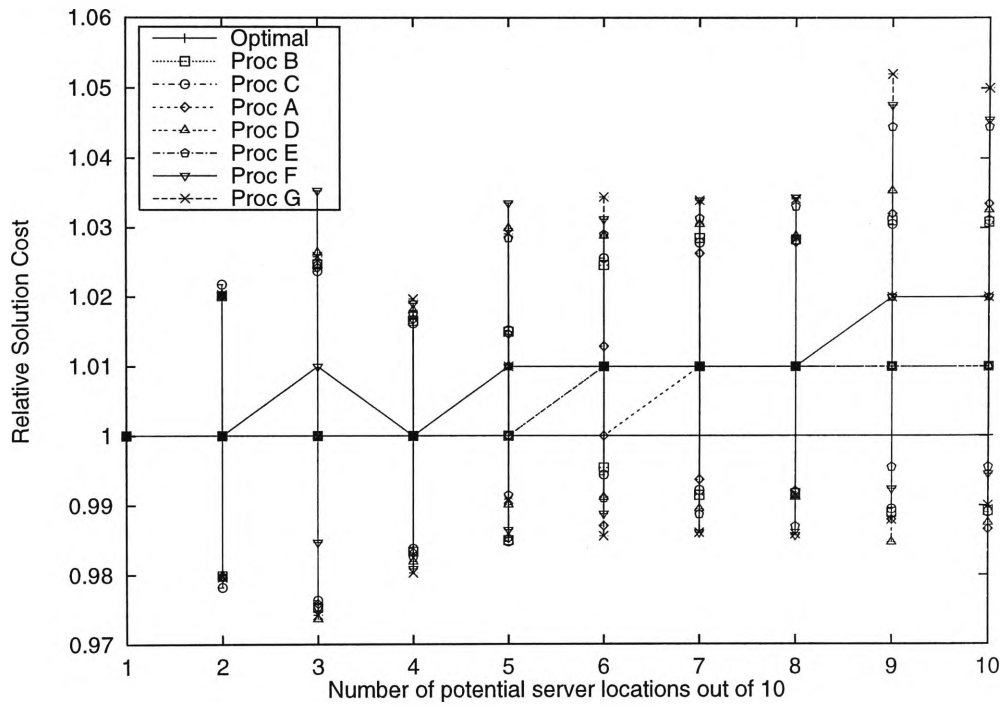


Figure 6.8 Relative cost of solutions produced by variations of the clustering procedure as the number of potential server locations varies. Network topologies are unconstrained.

It is interesting to note that as the number of potential server locations increases, the clustering procedures have more difficulty finding the optimal solution. Again, the faster clustering procedures are affected more than the slower, more complex versions. This is most likely because the faster procedures use more heuristics to reduce their execution time, while the slower procedures make fewer assumptions (use fewer heuristics) and search for the best solution instead.

6.4.1.5 Clustering Procedures Comparison Conclusion

The results from a comprehensive performance comparison of the various clustering procedures under a variety of conditions show that there is little (less than 2%) dif-

ference between them. When there is a difference between the procedures, the faster, less complex versions of the clustering procedure consistently produce slightly higher cost solutions than the slower, more complex procedures. As mentioned previously, an important point to note is that the significant difference in execution time between the fast and slow procedures is not associated with a corresponding penalty in solution quality. That is the execution time benefits of the faster procedures will often outweigh the slight penalty in solution cost. From a network designer's point of view the faster procedures could be used in the initial stages of a project to rapidly gain an understanding of the impact of various design parameters on the cost and hence feasibility of a project. Once the design parameters have been finalised all of the design procedures could be employed and the solutions compared. However, for larger design problems (e.g. more than 50 potential server locations) it may be possible to employ only the fastest variations of the clustering algorithm. The various forms of the clustering procedure allow the designer to chose a variation of the algorithm most appropriate for the current stage of the project.

To simplify the presentation of results in the remainder of this section we only include clustering procedures **A** and **G**, labelled as "slow" and "fast" respectively.

6.4.2 Clustering vs. Greedy Algorithms

In the previous section we compared the performance of variations of the node clustering based DSN design procedure. In this section we use the same experiments to compare two versions of the clustering procedure with both the ADD/DROP and ADD-*k* greedy search based DSN design procedures. In addition, in cases where the problem size makes it infeasible to determine the optimal solution, we employ the lower bound(s) produced by the continuous branch-and-bound procedure described in Chapter 5.

As in the previous section, the error-bars in all experiments represent the 95% confidence interval for each estimate. Furthermore, in these experiments the link topology of the networks designed were unconstrained. The corresponding results for the same networks with constrained network topologies are given in Appendix B.

6.4.2.1 Network Size

Figure 6.9 compares the relative cost of designs produced by the clustering, ADD/DROP, and ADD-*k* procedures (see Table 5.1), normalised with respect to the solution produced by the "slow" clustering procedure (**A**), for a range of network sizes. The results for the ADD-*k* procedure are truncated due to the high execution time of the procedure. Despite that, it is clear that the two greedy design procedures perform very similarly, and have a slight (and increasing) advantage over the clustering procedures.

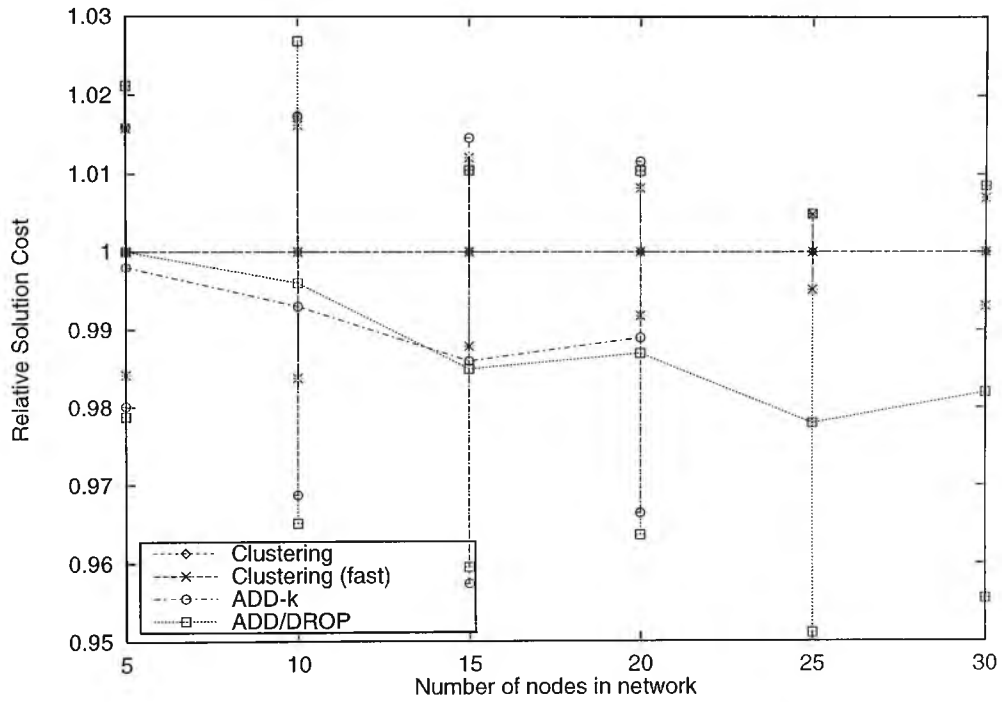


Figure 6.9 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-*k* procedures as the size of the network varies. Network topologies are unconstrained.

Figure 6.10 adds the lower bounds produced by the continuous branch-and-bound procedure (described in Section 5.5) to Figure 6.9. Recall that the branch-and-bound procedure operates by narrowing the gap between an upper and lower bound on the cost of the solution to the simplified DSNDP. Thus when the lower bounding procedure is unable to solve the simplified problem to optimality, it provides two lower bounds to the full DSNDP. From our 10 node network experiments (and 5 node network experiments, not included here), we know that the difference in cost between the solutions produced by the design procedures and the optimal solution is (on average) less than 2% in small networks. From this we can conclude that the 30-

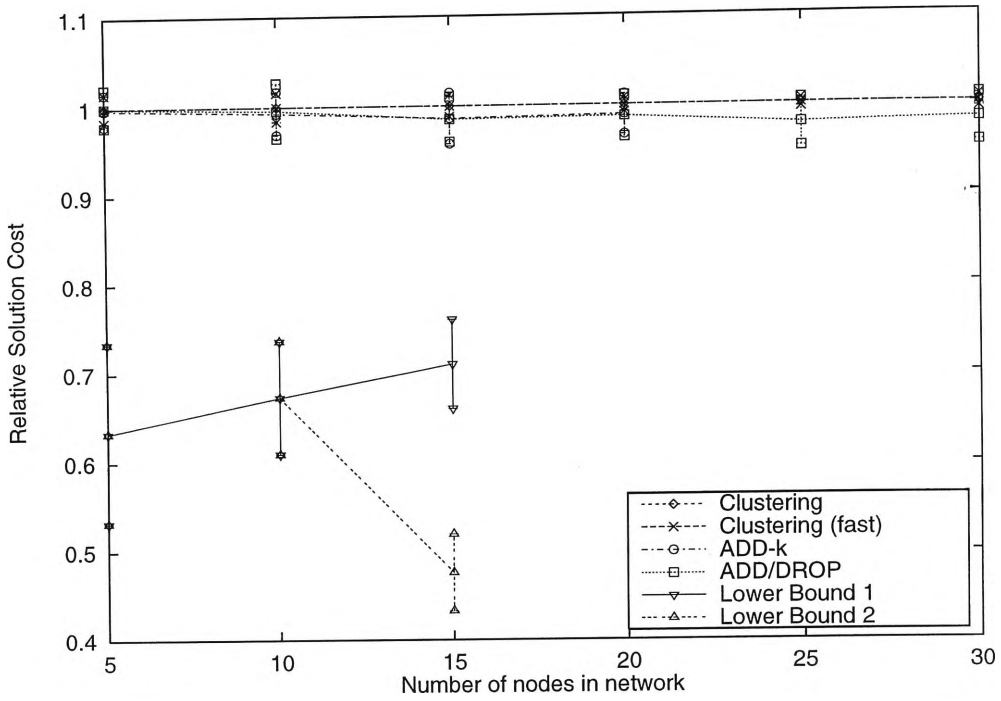


Figure 6.10 Relative cost of solutions produced by the clustering, ADD/DROP, ADD-*k*, and lower bounding procedures as the size of the network varies. Network topologies are unconstrained.

40% gap between the lower bound and design procedure results for 5 and 10 node networks is the minimum possible given the relaxed nature of the lower bounding problem. That is, we know that for small networks the design procedures are near-optimal, thus the 30-40% gap between them and the results from the relaxed lower-bounding problem must be almost entirely due to the relaxation rather than sub-optimality in the solutions found by the design procedures. We also know that for 15 node networks, the optimal solution to the lower bounding problem lies somewhere between the Lower Bound 1 and 2 curves due to the operation of the continuous branch-and-bound algorithm (see Section 4.4). As we have observed previously, experience (both our own and reported in the literature) suggests that branch-and-bound algorithms often find (near) optimal solutions very quickly, but take a long time to verify their optimality. This leads us to suggest that the optimum solution to the lower bounding problem lies closer to the Lower Bound 1 curve, rather than Lower Bound 2 curve. This means that, because the gap between the heuristic solution costs and the expected location of the optimal solution to the simplified lower bounding problem remains around 30% to 40% as the network size (i.e. number of

potential server locations) increases, we can conclude that the performance of all the heuristic design procedures remains close to optimal as the network size increases.

6.4.2.2 Server to Link Cost Ratio

Figure 6.11 shows the relative (to the "slow" clustering procedure) cost of solutions produced by the clustering, ADD/DROP, and ADD-*k* DSN design procedures as the relative cost of servers and links varies. 100 randomly generated 10 node DSNDP's, in which all nodes were considered as potential server locations, were solved for each point, and the error-bars represent the 95% confidence interval for the results. Again, the clustering procedures do not perform as well as the ADD/DROP or ADD-*k* procedures. However, as servers become more expensive relative to links, the optimal solutions employ fewer servers and all procedures consistently identify the optimal solution.

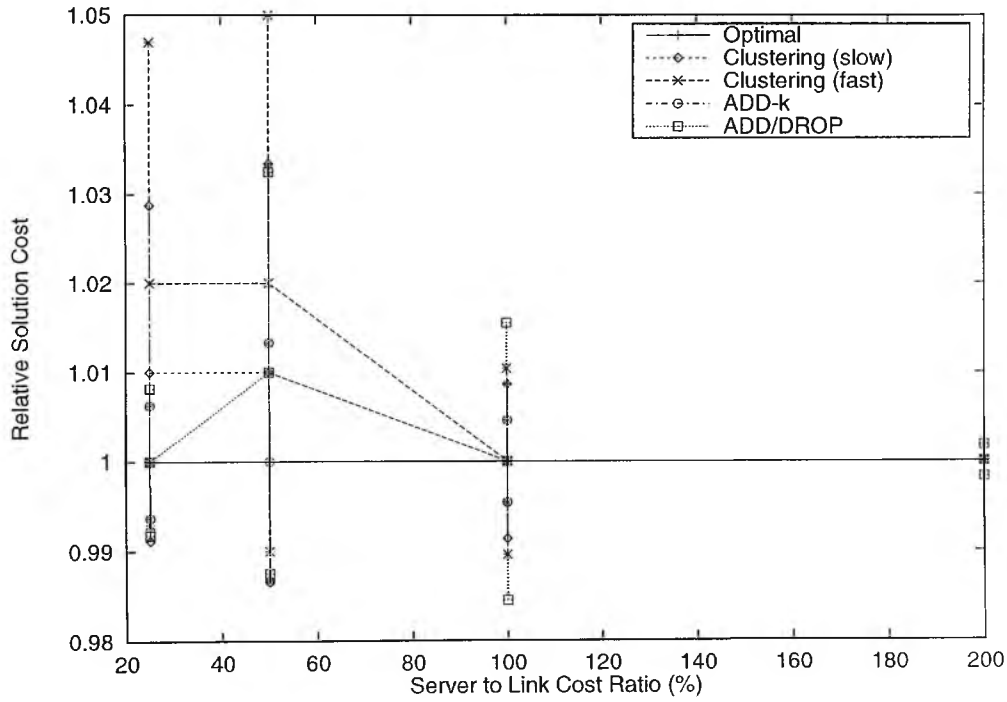


Figure 6.11 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-*k* procedures as the relative cost of servers and links varies. Network topologies are unconstrained.

6.4.2.3 Inter-server to Client-server Traffic Ratio

Figure 6.12 shows the cost of solutions produced by the clustering, ADD/DROP, and ADD-*k* procedures relative to the cost of the optimal solution, found via an

exhaustive search, as the ratio of inter-server to client-server traffic varies from 0% to 40%. As above, 100 randomly generated 10 node DSNDP's, in which all nodes were considered as potential server locations, were solved for each point, and the error-bars represent the 95% confidence interval for the results. Again, we see similar results as before. The ADD-*k* procedure consistently performs the best, followed by the ADD/DROP procedure, followed by the slow and then fast clustering procedures. Like the clustering procedures the ADD/DROP procedure is most easily able to identify the optimal solution when the volume of inter-server traffic is either zero or very high. The ability of the ADD-*k* procedure to effectively "look over" ridges in the solution space enables it to escape local minima and identify the optimal solution more often than the other procedures.

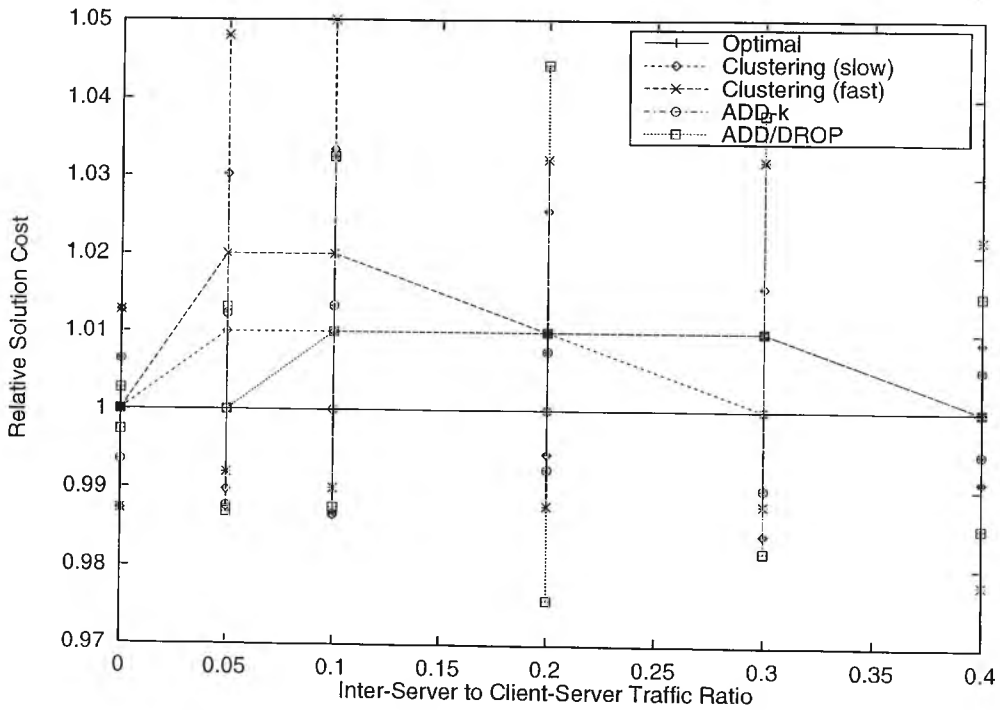


Figure 6.12 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-*k* procedures as the ratio of inter-server to client-server traffic varies. Network topologies are unconstrained.

Figure 6.13 adds the lower bounds produced by the continuous branch-and-bound procedure (described in Section 5.5) to Figure 6.12. The most significant relaxation of the DSNDP to obtain the lower bounding problem is the removal of the inter-server traffic. This means that when there is no inter-server traffic in the network the heuristic design procedures and the lower bounding procedure are effectively attempting to solve the same problem. Thus it is encouraging that there is almost no

difference between the cost of solutions produced by the heuristic design procedures and the lower bounding procedure when there is no inter-server traffic. Observe also, that the gap between the optimal solution (and heuristic design procedures) lower bound rapidly increases to, and stays at, approximately 30% to 40%, as inter-server traffic is introduced into the network. This lends weight to our earlier suggestion that the minimum gap we can expect between the lower bound and optimal solution to the complete DSNDP is around 30% to 40%.

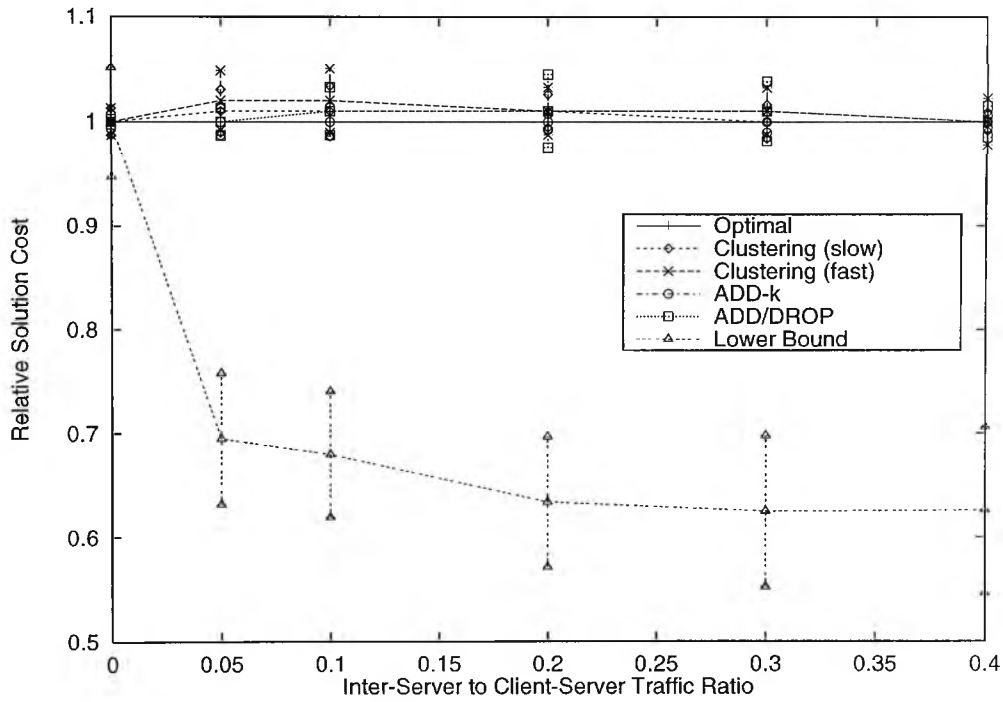


Figure 6.13 Relative cost of solutions produced by the clustering, ADD/DROP, ADD-*k*, and lower bounding procedures as the ratio of inter-server to client-server traffic varies. Network topologies are unconstrained.

6.4.2.4 The Number of Potential Server Locations Relative to the Total Number of Nodes in the Network

Figure 6.14 shows the cost of solutions produced by the clustering, ADD/DROP, and ADD-*k* procedures, relative to the optimal solution determined via exhaustive search, as the number of potential server locations varies from 1 to 10 in 10 node networks. The potential server locations were again selected as described in Section 6.4.1.4. These results show that the ADD-*k* procedure is robust, consistently being able to identify the optimal solution under a wide variety of conditions. The ADD/DROP procedure does almost as well, followed by the slow and fast clustering procedures.

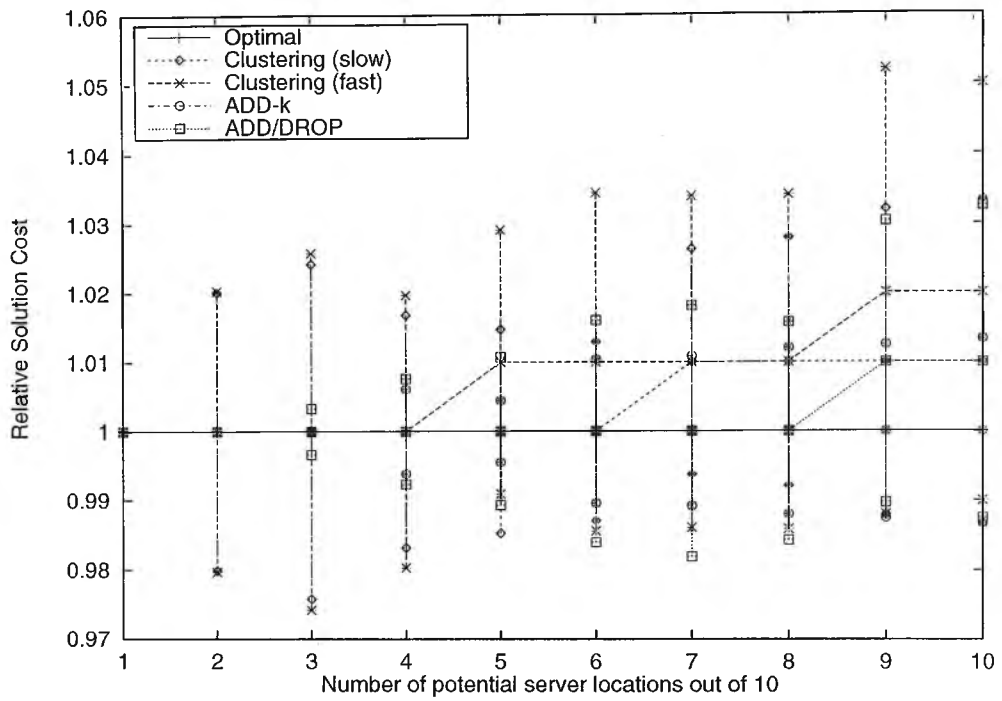


Figure 6.14 Relative cost of solutions produced by the clustering, ADD/DROP, and Add-*k* procedures as the number of potential server locations varies. Network topologies are unconstrained.

6.5 Clustering vs. Greedy Procedure Comparison Conclusions

In Section 6.2 we described a methodology that would aid network designers in applying the design procedures described in this dissertation. The key feature of all the DSN design algorithms is the ability to generate a complete description of the traffic requirements between all nodes in the network once the location of servers and assignment of client nodes to them is known. Combined with a set of link and server cost functions the link topology of the network can be designed and any potential solution evaluated.

There are a number of implementation options within the node clustering based design heuristic. The results in Section 6.4.1 and Appendix A show that there is little difference in the cost of the solutions produced by the various forms of the clustering procedure. However, the implementation options selected can have a significant effect on the execution time of the procedure.

When comparing the clustering and greedy DSN design procedures (see results in Sections 5.4, 6.4.2, and Appendix B), the ADD-*k* procedure consistently produces the lowest cost designs, followed by the ADD/DROP procedure, followed by the slow clustering and then fast clustering procedures. However, the difference in the cost of the solutions produced is minimal (less than 2% on average), and all produce solutions within 2% of optimum for small networks (up to 10 potential server locations). The results from the lower bounding procedure suggest that the procedures continue to produce near optimal solutions as the network size increases. Despite the small difference in solution quality, of all the procedures, the ADD/DROP and ADD-*k* procedures suffer a significant execution time penalty over the clustering procedures. In fact the execution time of the ADD-*k* procedure makes it impractical for designing networks with more than 25 potential server locations. Thus the ADD/DROP procedure is the best compromise in execution time and solution cost.

The availability of a number design procedures has two advantages for the network designer:

Firstly it, allows the designer to chose a variation of the algorithm most appropriate for the current stage of the project. As mentioned previously, the faster clustering procedures could be employed in the initial stages of a project to rapidly gain an understanding of the impact of various design parameters on the cost and hence feasibility of a project. Once the design parameters have been finalised, all of the design procedures could be employed (if possible) and the solutions compared. However, the use of the slower design procedures may be infeasible for larger design problems (e.g. more than 25 potential server locations).

Secondly, the differences in the algorithms employed within the design procedures means that they provide controls for one another. That is, if it is possible to employ all of the procedures and they all produce very similar solutions (which our results show they usually do) the designer can be confident that they have a near optimal solution. If only one procedure is employed then the designer has no way of being sure that it is not getting stuck in a sub-optimal portion of solution space due to some feature of the current problem.

If a designer is forced to select only one procedure, our results suggest that the ADD/DROP procedure represents the best balance between execution time and

solution quality. The ADD/DROP procedure is significantly faster than the ADD-k procedure, hence it is possible to design relatively large networks with it. Despite its speed, the cost of the solutions the ADD/DROP procedure produces are generally very close to those of the ADD-k procedure, (which is too slow for use on larger problems). While the clustering procedures are faster than the ADD/DROP procedure, they are more likely to be misled by features of specific design problems. This is because more information about the expected form of the optimal solution is built into the clustering procedures in comparison to the greedy search based procedures.

To conclude, a designer will be best served by using a selection of the design procedures presented here at different stages of the design process. However, if one design procedure must be chosen over the others our analysis indicates that the ADD/DROP procedure performs best in a variety of network environments.

7. Conclusions

The end can not justify the means, for the simple and obvious reason that the means employed determine the nature of the ends produced.

- Aldous Huxley, Ends and Means (1937).

7.1 Introduction

Distributed Server Networks (DSN's) are becoming more common with increased use of network services, such as the World Wide Web, and distributed databases providing, for example, banking services. Furthermore performance and/or reliability concerns will increasingly lead to multiple servers being used to implement such network services.

Designing DSN's is complex due to three main features. Firstly, the number of outputs required of the design process is large, these include the number, location and capacity of server, the allocation of clients to servers, and the design of the network interconnecting client to servers and servers to one another. As a result it combines aspects from many traditional or "standard" network design problems (see Chapter 2) into a single complex design problem. Secondly, the DSNDP includes concave cost functions to model the economies of scale that exist in the pricing of network links and servers. Finally, the trend to move away from traditional networks in which the possible link capacities are limited, to network technologies such as ATM where link capacities are much more flexible, complicate the problem by introducing another dimension to the design problem.

While there is much previous work in the area of network design, the DSNDP has not been addressed in its entirety. Standard network design problems, and their associated solution procedures do not include all aspects of the design of a DSN. As

shown in Chapter 3, some researchers have attempted to solve problems that, like the DSNDP, combine aspects of a number of the standard problems. However, this previous work does not provide a practical solution procedure for the DSNDP. Also in Chapter 3 we showed how the design problem can be viewed as a combinatorial optimisation problem in two parts - the first part being the generation of potential solutions, and the second, the evaluation of each potential solution. In this dissertation we have concentrated on these two parts of the design problem, neither of which, as shown in Chapters 2 and 3, is adequately addressed in the literature.

Firstly, to solve the DSNDP we also be able to find the optimal (or a near optimal) configuration of servers and client assignment for any given DSN by solving a facility location-allocation problem in the context of a DSN. That is, determining the number, and location of interacting servers, and the assignment of client nodes to them.

Secondly, to evaluate a DSN design, that is, a given configuration of servers and assignment of client nodes to them we must design a network to support their interaction. In the context of strongly concave link cost functions (the common case for the DSNDP) we must solve a Concave Topology, Capacity and Flow Assignment (TCFA) problem.

Our results and recommendations are described in the following sections. This dissertation concludes with a discussion of possibilities for further work in this area.

7.2 The Concave TCFA Problem

All three of the DSN design procedures presented in this dissertation require the solution of Concave TCFA problems. Initially only the traffic requirements of each client node are known in the DSNDP. However, once a set of servers is located and client nodes assigned to them a complete description of the required traffic flow between all node pairs can be generated. Due to the concave link cost functions included in the DSNDP designing the link topology to support those traffic requirements is a Concave TCFA problem.

Our review of previous work on backbone network design in Chapter 2 showed that there are only three existing network design procedures capable of solving the Concave TCFA problem. These are Kleinrock and Gerla's Concave Branch Elimination (CBE) procedure [Klei76] [Gerl77], Gavish *et al.*'s MILP procedure [Gavi86b] [Gavi89] [Gavi90] [Gavi92a], and Gersht and Weihmayer's greedy link elimination procedure [Gers90]. However, our experiments have shown that the CBE procedure does not perform well when link cost functions are strongly concave, a common occurrence in a DSN environment. Gavish *et al.*'s MILP procedure uses an exhaustive search to determine the capacity of each link. While this works fine when the number of possible link capacities is small, it will not perform well when the number of possible link capacities is large. Finally, the execution time of Gersht and Weihmayer's greedy link elimination procedure makes it useful for small network design problems only.

To rapidly provide a near optimal solution to the Concave TCFA problem we developed a Concave Link Elimination (CLE) procedure based on Gersht and Weihmayer's greedy link elimination procedure. The CLE procedure along with an extensive comparison of its performance with both the CBE and Gersht procedures was presented in Chapter 4. This comparison showed that the CLE procedure produces solutions whose cost is within 1% (on average) of Gersht's procedure, in significantly less time. For example, the CLE procedure takes two orders of magnitude less time than Gersht's procedure to design a 20 node network. When compared to the CBE procedure the cost of solutions produced by the CLE procedure are (on average) 45% cheaper than those of the CBE procedure. Moreover, for networks of up to 50 nodes the CLE procedure is also faster than the CBE procedure. Thus this dissertation describes a new technique (our CLE procedure) for solving Concave TCFA problems that significantly out-performs the existing procedures.

In addition to comparing the three TCFA procedures to one another we compared their performance to a lower bound. This provides not only an assessment of their relative performance, but also an assessment of how far the results they produce are from optimal. Thus a designer can be confident not only that they have the best procedure available, but also that the results are near optimal. The lower bounding problem was produced by relaxing the full Concave TCFA design problem. This lower bounding problem was solved using a procedure based on a continuous

branch-and-bound algorithm combined with Sequential Quadratic Programming (SQP). Our results show that the two greedy link elimination procedures (ours and Gersht's) consistently produced near optimal solutions. In contrast, the CBE procedure's performance degrades with the size of the network. To conclude, our CLE procedure provides near optimal solutions to the Concave TCFA problem of comparable quality to the best existing procedures in significantly less time.

7.3 The DSN Location-Allocation Problem

In broad terms a DSN consists of a number of client nodes communicating with a number of servers which in turn communicate with one another to provide the desired services to the client nodes. Any design procedure for a DSN produces a number of outputs including the number of servers to use, their location and how client nodes should be assigned to them. This type of problem is known as a facility location-allocation problem.

In Chapters 2 and 3 we reviewed the existing work on various styles of location-allocation problems. While that review shows that a great deal of work has been done in this area, it also showed that none of the existing location-allocation solution procedures is able to handle the type of location-allocation problem that occurs in the context of designing DSN's. The existing solution procedures often assume and rely on some feature, such as linear or non-traffic dependent cost functions, or a limited number of possible link capacities, that are not present in the DSNDP. Alternately they ignore one or more crucial aspects of the DSNDP, such as, the interaction between not only clients and servers but also between servers themselves and its cost, link and/or server capacities, the determination of the number of servers. Finally, many facility location solution procedures rely on search based algorithms that explore a large portion of the solution space. This is an approach that is not directly applicable to the DSNDP because (as we discuss in Chapter 3) the evaluation of a solution to the DSNDP is an expensive process. Hence, although various facility location-allocation problems have been considered in the literature, none of the existing solution procedures are directly applicable to the DSNDP.

In Chapter 5 we developed three heuristic design procedures to solve the DSN location-allocation problem. The first procedure, presented in Section 5.2.3, uses a node clustering approach and based on a procedure from [Mukh93a] and [Saha95]. The other two procedures are based on greedy searches of the possible solution space, an ADD/DROP procedure and an ADD-k procedure adapted from [Gavi92a]. See Sections 5.3.1 and 5.3.2 respectively for details. While the clustering procedure assumes that a cluster based solution will be optimal and tries to find it, the greedy search based procedures do not rely on any such assumptions and are free to search the solution space without such limitations. All three procedures rely on the CLE procedure from Chapter 4 and they use it repeatedly to evaluate the quality of each possible solution they generate.

In Section 5.4 we compared the complexity and execution time of the ADD/DROP, ADD-k and two variations of the clustering procedure. Our results, combined with those in Section 6.4.1 show that the options chosen in the clustering procedure significantly impact its speed of execution without significantly affecting the quality of the solutions produced. Our results also show that the execution time of the ADD/DROP procedure is approximately an order of magnitude slower than the faster clustering procedure for larger networks. Finally, the ADD-k ($k_{max} = 2$) procedure is an order of magnitude slower again to design a network with 20 potential server locations (the difference increases as k_{max} is increased). In addition, the difference between the execution times of the procedures increases with the number of potential server locations, or network size, to the extent that the ADD-k procedure is impractical for use in designing DSN's with more than 25 potential server locations. To summarise, our analysis showed that the faster node clustering procedure produced solutions to the DSN facility location-allocation problem in significantly less time that the other procedures.

To evaluate the quality of the solutions produced by our design procedures we describe, in Section 5.5.1, how a DSNDP can be relaxed and reformulated to define a less complex problem whose solution will be a lower bound on the cost of the optimum solution to the original problem. In Section 5.5.2 we describe how the continuous branch-and-bound algorithm used to provide a lower bounding procedure for the Concave TCFA problem in Chapter 4, can also be used to provide a lower

bounding procedure for the complete DSNDP. In addition, through experimentation we have found a value for the branch-and-bound procedure tolerance factor, ε, which reduces the execution time by an order of magnitude without significantly affecting solution quality.

An extensive performance comparison of our heuristic design procedures relative to one another and lower bounds produced by our procedure from Section 5.5.2 are presented in Sections 5.4, 6.4.2, and Appendix B. When comparing the clustering and greedy search based DSN design procedures, the ADD-*k* procedure consistently produces the lowest cost designs, followed by the ADD/DROP procedure, followed by the slow clustering and then fast clustering procedures. However, the difference in the cost of the solutions produced is minimal (less than 2% on average), and all produce solutions within 2% of optimum for small networks (up to 10 potential server locations). The results from the lower bounding procedure suggest that the procedures continue to produce near optimal solutions as the network size increases. Hence there is only a small difference in the quality of the solutions produced by the clustering and greedy search based procedures, with all producing near optimal solutions regardless of network size.

While there is little difference in the quality of the solutions produced by each of the procedures, as discussed above, they differ significantly in their execution times. The ADD/DROP and the ADD-*k* procedures suffer a significant execution time penalty over the clustering procedures. Our results lead us to conclude that the ADD/DROP procedure represents the best compromise in execution time and solution cost.

7.4 Aiding the DSN Researcher and Designer

The previous two sections discuss the development of a number of DSN design procedure, with results showing not only their relative performance, but also the trade-off between solution quality and the time required to produce a solution. In this dissertation we also attempt to aid designers in two significant ways. Firstly, we describe our own experimental design methodology which provides statistically sound measures of the relative performance of our design procedures. This same

experimental methodology could be applied to the assessment of any network design procedure. The key to our methodology is our ability to generate a large number of realistic "random" network design problems. Secondly, the DSN design procedures we have developed are intentionally very flexible and can be usefully applied to the DSN's with widely varying parameters. To facilitate the application of our DSN design procedures to practical problems we also provide a DSN design methodology that describes how all of design procedures can be used in concert to aid a designer in their task of designing a DSN.

Our procedure to generate "typical", or realistic, "random" example network design problems is described in detail in Section 4.7.1. As mentioned above, the example design problems produced by our procedure were used as the basis for the majority of the results presented in this dissertation and allow us to present statistically sound measurements of the relative performance of the our design procedures. As discussed in Section 3.6.1 the majority of previous work in network design either fails to provide any specific design examples or uses unrealistic examples in which the traffic requirements between all node pairs is uniform. While we are unable to comment on work in which no description of the example problems considered is provided, the design examples in which the traffic requirement are completely uniform traffic could lead to design procedures biased towards that sort of network. To avoid this risk in our work we tested our design procedures on a large number of "randomly" generated realistic network examples and used the average results, with confidence intervals, in all our comparisons. We also tested our design procedures by varying a number of the design parameters, thus investigating the sensitivity of our procedures to those design parameters. As described in Section 4.7.1 our typical network design procedures were generated by randomly distributing nodes about a geographic area, selecting a user population at each node from a Pareto distribution and then generating client-server traffic requirements based on their respective populations, proximity and traffic measurements taken from a live network. If an existing network topology was required we generated a peer-to-peer O-D traffic requirements matrix based on the same information and used the CLE procedure to design a link topology to support those requirements. Although to progress we had to select specific parameters for the Pareto distribution, and invent an equation to generate traffic requirements based on those populations, the specific details of

those processes are not significant. The aim of our work in this area was to ensure that we were able to produce large numbers of essentially random, yet typical network design problems with which to rigorously test our design procedures. This feature has, to our knowledge, been lacking from the existing network design research.

Finally, in Section 6.2 we tie all our work in this area into a DSN design methodology, which aims to encourage the practical application of the work presented in this dissertation and supporting publications. Furthermore, Sections 6.2.4 and 7.5 we discuss both how our procedures could be modified to cope with a variety of design requirements that we have not considered in detail but that could easily occur in practice, we also outline areas for further research into the design of DSN's that are beyond the scope of this dissertation.

7.5 Areas for Further Work

As discussed in Chapters 1, 3 and 6 we have made a number of assumptions for practical purposes that limit the scope of the DSNDP we have considered.

In Section 6.2.4 we discussed how some practical variations of the DSNDP that might be included in our design methodology, and how our design procedures could be enhanced to allow for them. Examples include:

- Designing a DSN in the presence of other traffic or services sharing the same network infrastructure. That is, not using Virtual Service Networks (VSN's) to protect the DSN from the impact of traffic produced by other services.
- Designing a multi-service DSN. That is, a DSN where each client node may connect to multiple servers each offering a different service.
- Designing a DSN where client nodes are connected to multiple servers to satisfy performance or reliability constraints.
- Designing a hierarchical DSN in which clients connect to a first tier of servers which in turn connect to a second tier of servers and so on.

Another area for further research is considering how the DSN's produced by our procedures cope with conditions exceeding those they were designed for. For

instance increased demand, varying demands through the day, or the occurrence of overloads or failures. This could then lead to further research into how our DSN design procedures could be enhanced to produce designs better able to cope with these changing conditions. The possible future research in these areas is discussed further below.

7.5.1 DSN Stability Analysis

All of the networks we have designed thus far have been designed to perform within given limits under a load determined by inputs from the designer. An interesting area for further research is to investigate how well the designs handle overload conditions. Overload conditions may be due to a general increase in traffic requirements, or more localised due to link or server failures. Ideally the reliability constraints built into the designs would mean that there would be total loss of service in the event of failure and any overload would result in a gradual degradation in performance. However, without further research, possibly via simulation of specific network examples, we cannot tell at what overload level the network would suffer catastrophic failure and loss of service to customers. This is an area of primary concern to commercial operators, hence further research in this area in the context of DSN's would be valuable.

7.5.2 DSN Expansion

Even if the DSN's produced by our design procedures handle overloads gracefully at some point they may need to be expanded geographically or to add capacity. The most common approach to a requirement for additional capacity in practice is simply to measure the "hot spots" in the system and add as much capacity the budget allows. However, an interesting area for further research would be determine how best to expand an existing DSN geographically to include additional client and possible server nodes. The effectiveness of the design procedures will be determined by the freedom they provide to alter the existing network. They may also suffer from starting from what will most likely be a local minima in the new solution space. There will also always be a point at which the expansion required is so great that redesigning the network from scratch would be the best option. It would then be useful to have design procedures that could be used to help manage the change by

providing transitional designs that help evolve the network to its new configuration. Again, this is an area of great interest to commercial operators as they seek to expand their existing services and add new ones.

7.5.3 Multiperiod DSN Design

Another area of interest to commercial operators is multi-period network design. This may take the form of budgeting for expansion in the network from day one by considering how best to design a network that has to support a demand of X in year one, $X+Y$ in year two, and so on. Alternately, most networks are designed using "busy-hour" traffic demands to dimension them. In an environment where demand fluctuates significantly but predictably it may be possible to dimension the core network to support the "average" demand and include some form of dynamic capacity addition mechanism to cope with temporary peaks in demand. Dynamic capacity allocation through the inclusion of temporary leased lines has been studied in [LeBl90], amongst other places. However, the means for achieving this in the context of a DSN have yet to be determined.

References

[Ahuj93] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice Hall, Englewood Cliffs, New Jersey 07632, 1993.

[Ayki92] Turgut Aykin and Gary F. Brown. Interacting new facilities and location-allocation problems. *Transportation Science*, 26(3):212–222, August 1992.

[Ayki94] Turgut Aykin. Lagragian relaxation based approaches to capacitated hub-and-spoke network design problem. *European Journal of Operational Research*, 79:501–523, 1994.

[Ayki95] Turgut Aykin. The hub location and routing problem. *European Journal of Operational Research*, 83:200–219, 1995.

[Bala89] A. Balakrishnan, T. L. Magnanti, and R. T. Wong. A dual-ascent procedure for large-scale uncapacitated network design. *Operations Research*, 37(5):716–740, September 1989.

[Barc91] Jaime Barcelo, Elena Fernandez, and Kurt O. Jornsten. Computational results from a new Lagrangian relaxation algorithm for the capacitated plant location problem. *European Journal of Operational Research*, 53:38–45, 1991.

[Barn94] Scott A. Barnett, Chris H. E. Stacey, Gary J. Anido, H. W. Beadle, and Hugh Bradlow. A prototype information service architecture in a distributed ATM environment. In *Proceedings of Australian Telecommunication Networks and Applications Conference (ATNAC'94)*, December 1994.

[Beas93a] David Beasley, David R. Bull, and Ralph R. Martin. An overview of genetic algorithms: Part 1. fundamentals. *University Computing*, 15(2):58–69, 1993.

[Beas93b] J.E. Beasley. Lagrangian heuristics for location problems. *European Journal of Operational Research*, 65:383–399, 1993.

[Berk95] Michel Berkelaar. *lp_solve: A Public Domain MILP Solver*, 1995. Available from ftp://ftp.es.ele.tue.nl/pub/lpsolve/.

[Bert95] Dimitri Bertsekas. *Nonlinear Programming*. Athena Scientific, 1995.

[Boor77] Robert R. Boorstyn and Howard Frank. Large-scale network topological optimization. *IEEE Transactions on Communications*, COM-25(1):29–47, January 1977.

[Borc94] Brian Borchers and John E. Mitchell. An improved branch and bound algorithm for mixed integer nonlinear programs. *Computers and Operations Research*, 21(4):359–367, 1994.

[Bous91] Choaib Bousba and Laurence A. Wolsey. Finding minimum cost directed trees with demands and capacities. *Annals of Operations Research*, 33:285–303, 1991.

[Butt96] Milena Butto, Domenico Marocco, and Marco Quagliotti. Backbone dimensioning of a data packet switched network with high speed links. In *Proceedings of the 4th International Conference on Telecommunication Systems - Modelling and Analysis*, pages 553–559, March 1996.

[Cahn91] Robert S. Cahn, Pao-Chi Chang, Parviz Kermani, and Aaron Kershenbaum. Intrepid: An integrated network tool for routing, evaluation of performance, and interactive design. *IEEE Communications Magazine*, pages 40–47, July 1991.

[Cahn95] Robert S. Cahn. MENTour: An algorithm for designing reliable high-speed data networks. *Canadian Journal of Electrical and Computer Engineering*, 20(3):101-103, 1995.

[Cahn96] Robert S. Cahn. The network/server design problem. In *Proceeding of the 4th International Conference on Telecommunication Systems - Modelling and Analysis*, pages 67–78, March 1996.

[Card89] Richard H. Cardwell, Clyde L. Monma, and Tsong-Ho Wu. Computer-aided design procedures for survivable fiber optic networks. *IEEE Journal on Selected Areas in Communications*, 7(8):1188–1197, October 1989.

[Chen80] Peter Pin-Shan Chen and Jacob Akoka. Optimal design of distributed information systems. *IEEE Transactions on Computers*, C-29(12):1068–1080, December 1980.

[Chen92] Y.L. Chen and Y.H. Chin. Multicommodity network flows with safety considerations. *Operations Research*, 40(S1):S48–S55, January 1992.

[Chha92] Dilip Chhajed and Timothy J. Lowe. Locating facilities which interact: Some solvable cases. *Annals of Operations Research*, 40:101–124, 1992.

[Chu69] Wesley W. Chu. Optimal file allocation in a multiple computer system. *IEEE Transactions on Computers*, C-18(10):885–889, October 1969.

[Chun92] S. Chung, Y. Myung, and D. Tcha. Optimal design of a distributed network with a two-level hierarchical structure. *European Journal of Operations Research*, 62:105–115, 1992.

[Cole79] Guy B. Coleman and Harry C. Andrews. Image segmentation by clustering. *Proceedings of the IEEE*, 67(5):773–785, May 1979.

[Coop63] Leon Cooper. Location-allocation problems. *Operations Research*, 11:331–343, 1963.

[Crai93] Teodor Gabriel Crainic, Louis Delorme, and Pierre Dejax. A branch-and-bound method for multicommodity location with balancing requirements. *European Journal of Operational Research*, 65:368–382, 1993.

[Curr91] John Current and Hasan Pirkul. The hierarchical network design problem with transshipment facilities. *European Journal of Operational Research*, 52:338–347, 1991.

[Davi87] L. Davis, editor. *Genetic Algorithms and Simulated Annealing*. Morgan Kauffmann, 1987.

[Devo82] Jay L. Devore. *Probability and Statistics for Engineering and the Sciences*. Brooks/Cole Publishing Co., Monterey, CA, USA, 1982.

[Diri77] Hudai Dirilten and Robert W. Donaldson. Topological design of distributed data communication networks using linear regression clustering. *IEEE Transactions on Communications*, COM-25(10):1083–1092, October 1977.

[Dutt92] Amitava Dutta and Jay-Ick Lim. A multiperiod capacity planning model for backbone computer communication networks. *Operations Research*, 40(4):689–705, July 1992.

[Dysa78] Hugh G. Dysart and Nicolaos D. Georganas. NEWCLUST: An algorithm for the topological design of two-level, multidrop teleprocessing networks. *IEEE Transactions on Communications*, COM-26(1):55–62, January 1978.

[Efro66] M. A. Efroymson and T. L. Ray. A branch-bound algorithm for plant location. *Operations Research*, 14:361–368, May 1966.

[Erle78] Donald Erlenkotter. A dual-based procedure for uncapacitated facility location. *Operations Research*, 26(6):992–1009, November 1978.

[Fish81] Marshall L. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management Science*, 27(1):1–18, January 1981.

[Fish85] Marshall L. Fisher. An applications oriented guide to lagrangian relaxation. *Interfaces*, 15(2):10–21, March 1985.

[Fran72] Howard Frank and Wushow Chou. Topological optimization of computer networks. *Proceedings of the IEEE*, 60(11):1385–1937, November 1972.

[Frat72] L. Fratta, Mario Gerla, and Leonard Kleinrock. The flow deviation method: An approach to store-and-forward computer-communication network design. *Networks*, 3:97–133, 1972.

[Galv80] Roberto D. Galvao. A dual-bounded algorithm for the p-median problem. *Operations Research*, 28(5):1112–1121, September 1980.

[Gao94] Li-Lian Gao and E. Powell Robinson Jr. Uncapacitated facility location: General solution procedure and computational experinence. *European Journal of Operational Research*, 76:410–427, 1994.

[Gare79] Micheal R. Garey and David S. Johnson. *Computers and Intractability A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[Gavi82] Bezalel Gavish. Topological design of centralized computer networks - formulations and algorithms. *Networks*, 12:355–377, 1982.

[Gavi85a] Bezalel Gavish. Augmented lagrangian based algorithm for centralized network design. *IEEE Transactions on Communications*, COM-33(12):1247–1257, December 1985.

[Gavi85b] Bezalel Gavish. Models for configuring large-scale distributed computing systems. *AT&T Technical Journal*, 64(2):491, February 1985.

[Gavi86a] Bezalel Gavish. Computer and database location in distributed computer systems. *IEEE Transactions on Computers*, C-35(7):583–590, July 1986.

[Gavi86b] Bezalel Gavish and Irina Neuman. Capacity and flow assignment in large computer networks. In *Proceedings of IEEE Infocom'86*, pages 275–284. IEEE, April 1986.

[Gavi87] Bezalel Gavish. Optimization models for configuring distributed computer systems. *IEEE Transactions on Computers*, C-36(7):773–793, July 1987.

[Gavi89] Bezalel Gavish and Irina Neuman. A system for routing and capacity assignment in computer communication networks. *IEEE Transactions on Communications*, 37(4):360–366, April 1989.

[Gavi90] Bezalel Gavish and Kemal Altinkemer. Backbone network design tools with economic tradeoffs. *Operations Research Society of America Journal on Computing*, 2(3):236–247, 1990.

[Gavi91] Bezalel Gavish. Topological design of telecommunication networks - local access design methods. *Annals of Operations Research*, 33:17–71, 1991.

[Gavi92a] Bezalel Gavish. Topological design of computer communication networks - the overall design problem. *European Journal of Operational Research*, 58:149–172, 1992.

[Gavi92b] Bezalel Gavish, Chung-Lun Li, and David Simchi-Levi. Analysis of heuristics for the design of tree networks. *Annals of Operations Research*, 36:77–86, 1992.

[Gavi92c] Bezalel Gavish and Myung W. Suh. Configuration of fully replicated distributed database system over wide area networks. *Annals of Operations Research*, 36:167–192, 1992.

[Gavi95] Bezalel Gavish and Suresh Sridhar. Computing the 2-median on tree networks in O(n lg n) time. *Networks*, 26:305–317, 1995.

[Geof74] A.M. Geoffrion. Lagrangian relaxation for integer programming. *Mathematical Programming Study 2*, pages 82–114, 1974.

[Gerl73] Mario Gerla. Deterministic and adaptive routing policies in packeting networks. In *ACM-IEEE 3rd Data Communication Symposium*, Tampa, Florida, 1973.

[Gerl77] Mario Gerla and Leonard Kleinrock. On the topological design of distributed computer networks. *IEEE Transactions on Communications*, Com-25(1):48–60, January 1977.

[Gers90] Alexander Gersht and Robert Weihmayer. Joint optimisation of data network design and facility selection. *IEEE Journal on Selected Areas in Communications*, 8(9):1667–1681, December 1990.

[Glov89] F. Glover. Tabu search, part i. *ORSA Journal on Computing*, 1(3):190–206, 1989.

[Glov90] F. Glover. Tabu search, part ii. *ORSA Journal on Computing*, 2(1):4–32, 1990.

[Gouv93] Luis Gouveia. A comparison of directed formulations for the capacitated minimal spanning tree problem. *Telecommunication Systems*, 1:51–76, 1993.

[Grou87] V.M. Grout, P.W. Sanders, and C.T. Stockel. Practical approach to the optimization of large-scale switching networks. *Computer Communications*, 10(1):30–38, February 1987.

[Haki64] S. L. Hakimi. Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations Research*, 12:450–459, 1964.

[Haki65] S.L. Hakimi. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations Research*, 13:462–475, 1965.

[Hand79] G. Y. Handler and P. B. Mirchandani. *Location on Networks - Theory and Applications*. MIT Press, Cambridge MA, 1979.

[Hors93] Reiner Horst and Hoang Tuy. *Global Optimization Deterministic Approaches*. Springer-Verlag, 1993.

[Huyn77] Dieu Huynh, Hisashi Kobayashi, and Franklin F. Kuo. Optimal design of mixed-media packet-switching networks: Routing and. *IEEE Transactions on Communications*, 25(1):158–169, January 1977.

[Jaer72] P. Jaervinen, J. Rajala, and H. Sinervo. A branch-and-bound algorithm for seeking the p-median. *Operations Research*, 20:173–178, 1972.

[Jook89] J. N. Jooker. Solving nonlinear multiple-facility network location problems. *Networks*, 19:117–133, 1989.

[Kami91] Kunio Kamimura and Hisakazu Nishino. An efficient method for determining economical configurations of elementary packet-switched networks. *IEEE Transactions on Communications*, 39(2):278–288, February 1991.

[Kell82] David L. Kelly and Basheer M. Khumawala. Capacitated warehouse location with concave costs. *Journal of the Operational Research Society*, 33:817–826, 1982.

[Kers80] A. Kershenbaum, R. Boorstyn, and R. Oppenheim. Second-order greedy algorithms for centralised teleprocessing network design. *IEEE Transactions on Communications*, COM-28(10):1835–38, October 1980.

[Kers91] Aaron Kershenbaum, Parviz Kermani, and George A. Grover. Mentor: An algorithm for mesh network topological optimization and routing. *IEEE Transactions on Communications*, 39(4):503–513, April 1991.

[Khum72] Basheer M. Khumawala. An efficient branch and bound algorithm for the warehouse location problem. *Management Science*, 18(12):B–718–B–731, August 1972.

[Khum74] Basheer M. Khumawala and David L. Kelly. Warehouse location with concave costs. *INFOR*, 12(1):55–65, February 1974.

[Kim95] Hyun-Joon Kim, Sung-Hak Chung, and Dong-Wan Tcha. Optimal design of the two-level distributed network with dual homing local connections. *IIE Transactions*, 27:555–563, 1995.

[Kim96] Min-Jeong Kim, Sang-Baeg Kim, and Kwan-Hong Ryu. The capacitated hierarchical p-median problem for facility locations of pcs networks. In *Proceedings of the 4th International Conference on Telecommunication Systems - Modelling and Analysis*, pages 223–228, March 1996.

[Kirk83] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–679, 1983.

[Klei76] Leonard Kleinrock. *Queueing Systems Volume 2: Computer Applications*, volume 2. John Wiley & Sons, Inc., 1976.

[Klei80] Leonard Kleinrock and Farouk Kamoun. Optimal clustering structures for hierarchical topological design of large computer networks. *Networks*, 10:221–248, 1980.

[Klin85] John G. Klincewicz. A large-scale distribution and location model. *AT&T Technical Journal*, 65(7):1705–1730, September 1985.

[Klin86] John G. Klincewicz and Hanan Luss. A lagrangian relaxation heuristic for capacitated facility location with single-source constraints. *Journal of Operational Research Society*, 37(5):495–500, 1986.

[Klin91] J.G. Klincewicz. Heuristics for the p-hub location problem. *European Journal of Operational Research*, 53:25–37, 1991.

[Kuro88] James F. Kurose and Hussein T. Mouftah. Computer-aided modeling, analysis and design of communication networks. *IEEE Journal on Selected Areas in Communications*, 6(1):130–45, January 1988.

[Lawr96] Craig Lawrence, Jian L. Zhou, and Andre L. Tits. *User's Guide for CFSQP Version 2.4: A C Code for Solving (Large Scale) Constrained Nonlinear (Minimax) Optimization Problems, Generating Iterates Satisfying All Inequality Constraints*. Electrical Engineering Department and Institute for Systems Research, University of Maryland, February 1996. Enquires should be directed to Prof. Andre L. Tits, E-mail: andre@eng.umd.edu.

[LeBl90] Larry LeBlanc. Design and operation of packet-switched networks with uncertain message requirements. *IEEE Transactions on Communications*, 38(8):1223–30, August 1990.

[Lee95] Choong Y. Lee. Application of a cross decomposition algorithm to a location and allocation problem in distributed systems. *Computer Communications*, 18(5):367–377, May 1995.

[Lee96] Kyungsik Lee, Kyungchul Park, and Sungsoo Park. Design of capacitated networks with tree configurations. *Telecommunication Systems*, 6:1–19, 1996.

[Levi78] K. Dan Levin and Howard Lee Morgan. A dynamic optimization model for distributed databases. *Operations Research*, 26(5):824–835, September 1978.

[Lo89] Chi-Chun Lo and Aaron Kershenbaum. A two-phase algorithm and performance bounds for the star-star concentrator location problem. *IEEE Transactions on Communications*, 37(11):1151–63, November 1989.

[Louv92] Francois V. Louveaux and D. Peeters. A dual-based procedure for stochastic facility location. *Operations Research*, 40(3):564–573, May 1992.

[Luen89] David G. Luenberger. *Linear and Non-Linear Programming*. Addison-Wesley, 2nd edition, 1989.

[Magn84] T. L. Magnanti and R. T. Wong. Network design and transportation planning: Models and algorithms. *Transportation Science*, 18(1):1–55, February 1984.

[Mate96] Geraldo R. Mateus and Raphael Valery L. Franqueira. Model and heuristic for a generalized access network design problem. In *Proceedings of the 4th International Conference on Telecommunication Systems - Modelling and Analysis*, pages 457–463, March 1996.

[McGi94] J. McGibney, D.D. Botvich, and T. Curran. Modern global optimisation heuristics in the long term planning of networks. In *Proceedings of the Australian Telecommunications Networks and Applications Conference*, pages 317–22, Melbourne, December 1994.

[McGr77] Patrick V. McGregor and Diana Shen. Network design: An algorithm for the access facility location problem. *IEEE Transactions on Communications*, COM-25(1):61–73, January 1977.

[Mino89] M. Minoux. Network synthesis and optimum network design problems: Models, solution methods and applications. *Network*, 19:313–360, 1989.

[Mirz85] Andranik Mirzaian. Lagrangian relaxation for the star-star concentrator location problem: Approximation algorithm and bounds. *Networks*, 15:1–20, 1985.

[Monm86] Clyde L. Monma and Diane D. Sheng. Backbone network design and performance analysis: A methodology for packet switching networks. *IEEE Journal on Selected Areas in Communications*, SAC-4(6):946–965, September 1986.

[Mukh93a] A. Mukherjee, B.B. Bhaumik, and A.K. Bandyopadhyay. An energy function approach and its application to a new clustering algorithm. In *Proceedings of the IEEE TENCON'93*, pages 877–881. Beijing, 1993.

[Mukh93b] A. Mukherjee, B.B. Bhaumik, and A.K. Bandyopadhyay. Hierarchical clustering technique for the minimum cost design of a naturally clustered computer communication network. In *Proceedings of the IEEE TENCON'93*, pages 476–478. Beijing, 1993.

[Nara90] Sridhar Narasimhan. The concentrator location problem with variable coverage. *Computer Networks and ISDN Systems*, 19:1–10, 1990.

[Nara94] Sridhar Narasimhan. Locating concentrators in a computer network with multiple coverage for terminals. *Computer Communications*, 17(2):94–102, February 1994.

[Naru77] S. C. Narula, U. I. Ogbu, and H. M. Samuelsson. An algorithm for the p-median problem. *Operations Research*, 25:709–713, 1977.

[Nemh88] George L. Nemhauser and Laurence A. Wolsey. *Integer and Combinatorial Optimization*. Wiley-Interscience Series. John Wiley & Sons, 1988.

[Newp89] Kris T. Newport and Pramod K. Varshney. On the design of performance-constrained survivable networks. In *IEEE Military Communications Conference (MILCOM'89)*, pages 38.1.1–8, New York, October 1989. IEEE.

[NSFNET95]NSFNET backbone traffic distribution by service, April 1995. This document is available at http://www.cc.gatech.edu/gvu/stats/NSF/9504.html

[Ng87] Tomy M.J. Ng and Doan B. Hoang. Joint optimization of capacity and flow assignment in a packet-switched communications network. *IEEE Transactions on Communications*, COM-35(2):202–209, February 1987.

[O'Ke86] Morton E. O'Kelly. The location of interacting hub facilities. *Transportation Science*, 20(2):92–106, May 1986.

[O'Ke87] Morton E. O'Kelly. A quadratic integer program for the location of interacting hub facilities. *European Journal of Operational Research*, 32:393–404, 1987.

[O'Ke92] Morton E. O'Kelly. A clustering approach to the planar hub location problem. *Annals of Operations Research*, 40:339–353, 1992.

[Ouve94] Iradj Ouveysi. A lagrangian relaxation technique for multi-connected terminals and concentrator location problem. In *Proceedings of the Australian Telecommunications Networks and Applications Conference*, pages 323–28, Melbourne, December 1994.

[Pirk86] Hasan Pirkul. An integer programming model for the allocation of databases in a distributed computer system. *European Journal of Operations Research*, 26:401–411, 1986.

[Pirk88] Hasan Pirkul, Sridhar Narasimhan, and Prabuddha De. Locating concentrators for primary and secondary coverage in a computer communications network. *IEEE Transactions on Communications*, 36(4):450–458, April 1988.

[Pirk89] Hasan Pirkul. The uncapacitated facility location problem with primary and secondary facility requirements. *IIE Transactions*, 21(4):337–348, December 1989.

[Pirk92] Hasan Pirkul and Vaidyanathan Nagarajan. Locating concentrators in centralized computer networks. *Annals of Operations Research*, 36:247–262, 1992.

[Pirk96] Hasan Pirkul and Rakesh Gupta. Centralized computer network topology design. In *Proceedings of the 4th International Conference on Telecommunication Systems - Modelling and Analysis*, pages 440–451, March 1996.

[Popp96] Fabrice Poppe and Piet Demeester. An integrated approach to a capacitated survivable network design problem. In *Proceedings of the 4th International Conference on Telecommunication Systems - Modelling and Analysis*, pages 391–397, March 1996.

[Reve70] Charles Revelle, David Marks, and Jon C. Liebman. An analysis of private and public sector location models. In *Management Science* [Robe71], pages 692–707. See also Robers (1971).

[Righ95] Giovanni Righini. A double annealing algoruithm for discrete location/allocation problems. *European Journal of Operational Research*, 86:452–468, 1995.

[Robe71] Philip D. Robers. Some comments concerning revelle, marks and liebman's article on facility location. In *Management Science* [Reve70], pages 109–111. See also Robers (1971).

[Ryoo95] H. S. Ryoo and N. V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers and Chemical Engineering*, 19(5):551–566, 1995.

[Saha95] D. Saha and A. Mukherjee. Design of hierarchical communication networks under node/link failure constraints. *Computer Communications*, 18(5):378–83, May 1995.

[Schn82] G. Michael Schneider and Mary N. Zastrow. An algorithm for the design of multilevel concentrator networks. *Computer Networks*, 6:1–11, 1982.

[SG96] Ernesto Santibanez-Gonzalez, Felipe Chamas, and Henrique Pacca. A tightest formulation for a local access network design problem. In *Proceedings of the 4th International Conference on Telecommunication Systems - Modelling and Analysis*, pages 452–456, March 1996.

[Shap79] Jeremy F. Shapiro. A survey of lagrangian techniques for discrete optimization. *Annals of Discrete Mathematics*, 5:113–138, 1979.

[Shar93] Nita Sharma and Dharma P. Agrawal. Hierarchical distributed system network design with cost-performance tradeoffs. In *Proceedings of the 2nd International Symposium on High Performance Distributed Computing*, pages 336–343, Los Alamitos, CA, USA, July 1993. IEEE Computer Society Press.

[SK94] Darko Skorin-Kapov and Jadranka Skorin-Kapov. On tabu search for location of interacting hub facilities. *European Journal of Operational Research*, 73:502–509, 1994.

[Sola74] Richard M. Soland. Optimal facility location with concave costs. *Operations Research*, 22:373–382, March 1974.

[Spie69] Kurt Spielberg. Algorithms for the simple plant-location problem with some side constraints. *Operations Research*, 17:85–111, 1969.

[Srid93] R. Sridharan. A Lagrangian heuristic for the capacitated plant location problem with single source constraints. *European Journal of Operational Research*, 66:305–312, 1993.

[Srid95] R. Sridharan. The capacitated plant location problem. *European Journal of Operational Research*, 87:203–213, 1995.

[Stac95] Chris H. E. Stacey, Gary J. Anido, H.W. Peter Beadle, and Hugh Bradlow. Multipoint, multimedia service network dimensioning. In *Proceedings on the Australian Telecommunication Networks and Applications Conference 1995 (ATNAC'95)*, pages 473–8, Sydney, December 1995. A Postscript version is available from http://www.elec.uow.edu.au/conferences/95-66.ps.

[Stac96a] Chris H. E. Stacey, Tony Eyers, and Gary J. Anido. Client/server network design. Technical Report TITR-96-1, The Institute for Telecommunications Research, University of Wollongong, July 1996. A Postscript version is available from http://www.snrc.uow.edu.au/.

[Stac96b] Chris H. E. Stacey, Tony Eyers, and Gary J. Anido. Network design in a hybrid client/server peer network environment. In *Proceedings of Networks'96*, Sydney, November 1996. A Postscript version is available from http://www.snrc.uow.edu.au/.

[Stac96c] Chris H. E. Stacey, Tony Eyers, and Gary J. Anido. A node clustering approach to the design of hybrid access/backbone networks. In *Proceedings on the Australian Telecommunication Networks and Applications Conference 1996 (ATNAC'96)*, Melbourne, December 1996. A Postscript version is available from http://www.snrc.uow.edu.au/.

[Stac97a] Chris H. E. Stacey, Tony Eyers, and Gary J. Anido. A concave, link elimination (CLE) procedure and lower bound for concave, topology, capacity and flow assignment (TCFA) network design problems. In *Proceedings of the International Conference on Telecommunication Systems Modelling and Analysis 1997 (ICTS97)*, Nashville, TN 37203, USA, March 1997. A Postscript version is available from http://www.snrc.uow.edu.au/.

[Stac97b] Chris H. E. Stacey, Tony Eyers, and Gary J. Anido. A concave, link elimination (CLE) procedure and lower bound for concave, topology, capacity and flow assignment (TCFA) network design problems. To appear in Telecommunications Systems journal.

[Stac97c] Chris H. E. Stacey, Tony Eyers, and Gary J. Anido. A lower bound for client-server network design problems. In *Proceedings of the International Conference on Telecommunications 1997 (ICT97)*, Melbourne, April 1997. A Postscript version is available from http://www.snrc.uow.edu.au/.

[Tang78] Donald T. Tang, Lin S. Woo, and Lalit R. Bahl. Optimization of teleprocessing networks with concentrators and multiconnected terminals. *IEEE Transactions on Computers*, C-27(7):594–604, July 1978.

[Yan95] James Yan and Maged Beshai. Designing an ATM-based broadband network: An overview. In *Proceedings of GLOBECOM'95*, pages 245–251, November 1995.

[Zhan96] Shuzhi Zhang, Griff L. Bilbro, and Salah E. Elmaghraby. ATM network topological design: Heuristic and lower bound. In *Proceedings of the 4th International Conference on Telecommunication Systems - Modelling and Analysis*, pages 275–284, March 1996.

Appendix A. Additional Results from a Performance Comparison of Variations of the Node Clustering DSN Design Algorithm

In Section 6.4.1 the performance of variations of the node clustering based DSN design procedures were compared. The results included in Section 6.4.1 were from experiments in which the link topologies of the networks were unconstrained. In this appendix we include the corresponding results when network link topologies were constrained. As discussed in Section 6.2.3 the CLE procedure was used to design a backbone link topology for each network. The DSN design is constrained to using only those links included in the CLE design. The results in this appendix differ very little from those in Section 6.4.1, they are included here for completeness. Hence only the graphs of the results are presented here. The analysis of the graphs in Chapter 6 is directly applicable to the results presented here as described below.

Figures A.1 and A.2 correspond to Figures 6.4 and 6.5 respectively, the reader is referred to Section 6.4.1.1 for further details.

Figure A.3 corresponds to Figure 6.6, the reader is referred to Section 6.4.1.2 for further details.

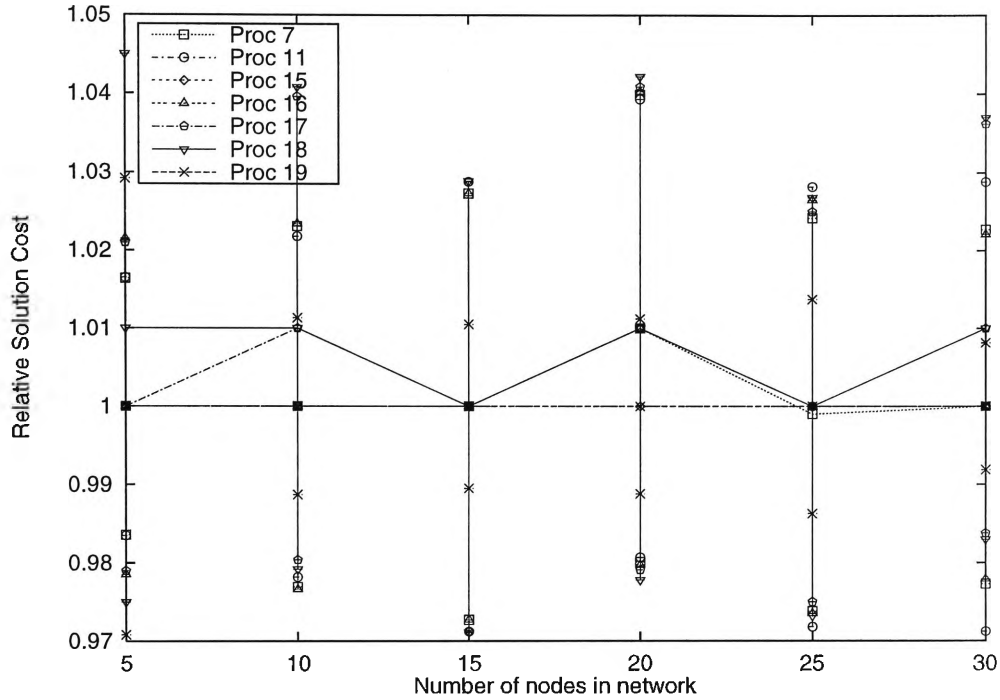


Figure A.1 Relative cost of solutions produced by variations of the clustering procedure as the size of the network varies. Network topologies are constrained.

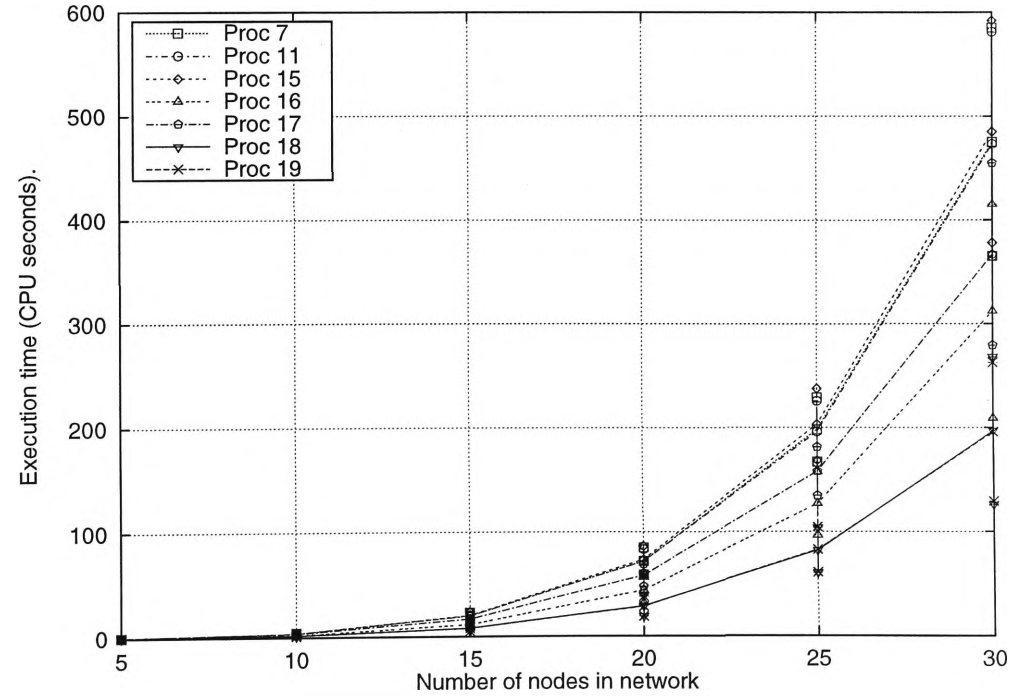


Figure A.2 Execution time of variations of the clustering procedure as the size of the network varies. Network topologies are constrained.

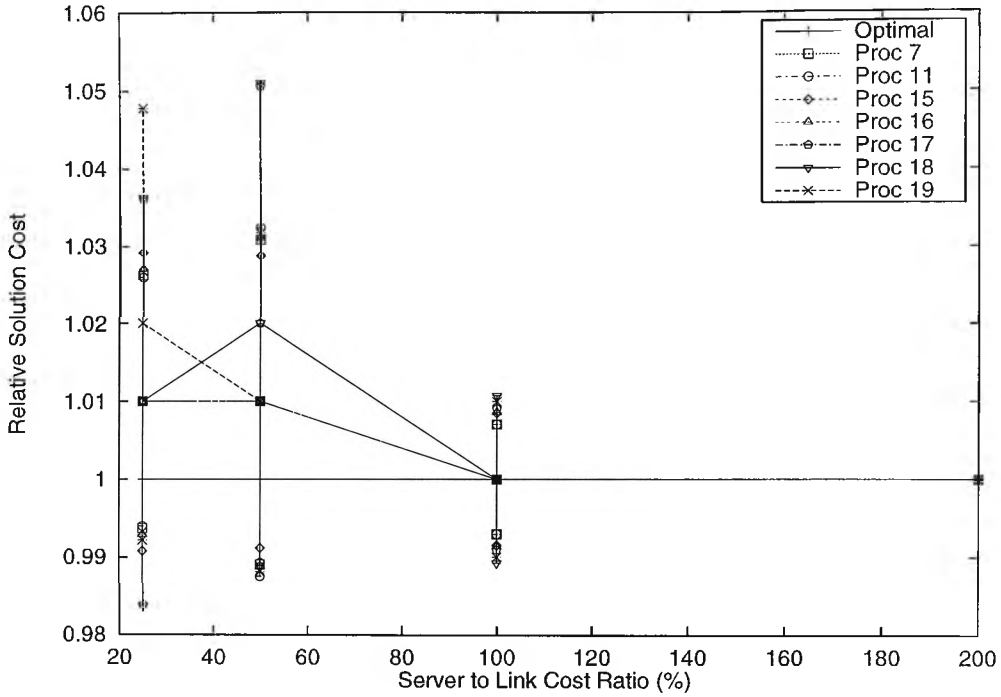


Figure A.3 Relative cost of solutions produced by variations of the clustering procedure as the relative cost of servers and links varies. Network topologies are constrained.

Figure A.4 corresponds to Figure 6.7, the reader is referred to Section 6.4.1.3 for further details.

Figure A.5 corresponds to Figure 6.8, the reader is referred to Section 6.4.1.4 for further details.

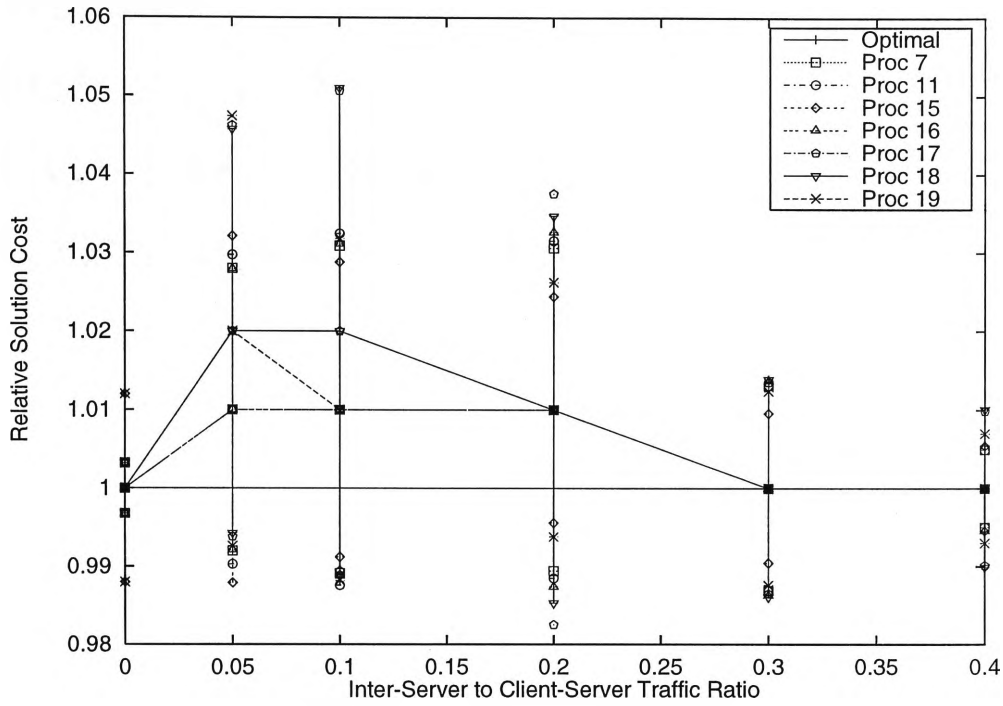


Figure A.4 Relative cost of solutions produced by variations of the clustering procedure as the ratio of inter-server to client-server traffic varies. Network topologies are constrained.

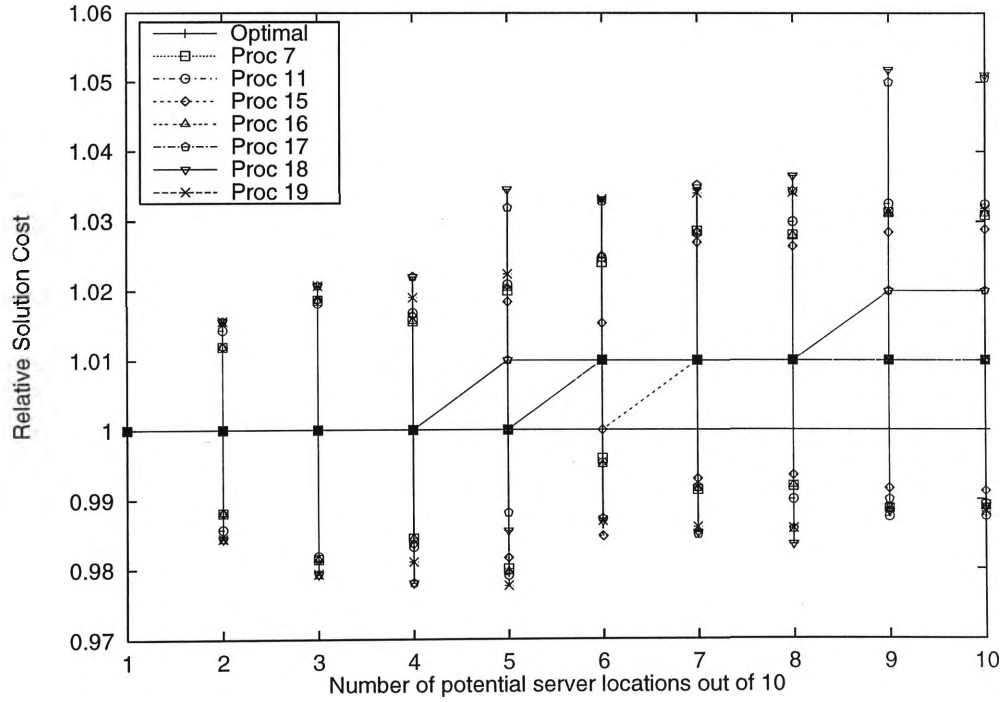


Figure A.5 Relative cost of solutions produced by variations of the clustering procedure as the number of potential server locations varies. Network topologies are constrained.

Appendix B. Additional Results from a Performance Comparison of the Node Clustering, ADD/DROP, and ADD-*k* DSN Design Procedures

In Section 6.4.2 the performance of the clustering, ADD/DROP, and ADD-*k* DSN design procedures were compared. The results included in Section 6.4.2 were from experiments in which the link topologies of the networks were unconstrained. In this appendix we include the corresponding results when network link topologies were constrained. As discussed in Section 6.2.3 the CLE procedure was used to design a backbone link topology for each network. The DSN design is constrained to using only those links included in the CLE design. The results in this appendix differ very little from those in Section 6.4.2, they are included here for completeness. Hence only the graphs of the results are presented here. The analysis of the graphs in Chapter 6 is directly applicable to the results presented here as described below.

Figures B.1 and B.2 correspond to Figures 6.9 and 6.10 respectively, the reader is referred to Section 6.4.2.1 for further details.

Figure B.3 corresponds to Figure 6.11, the reader is referred to Section 6.4.2.2 for further details.

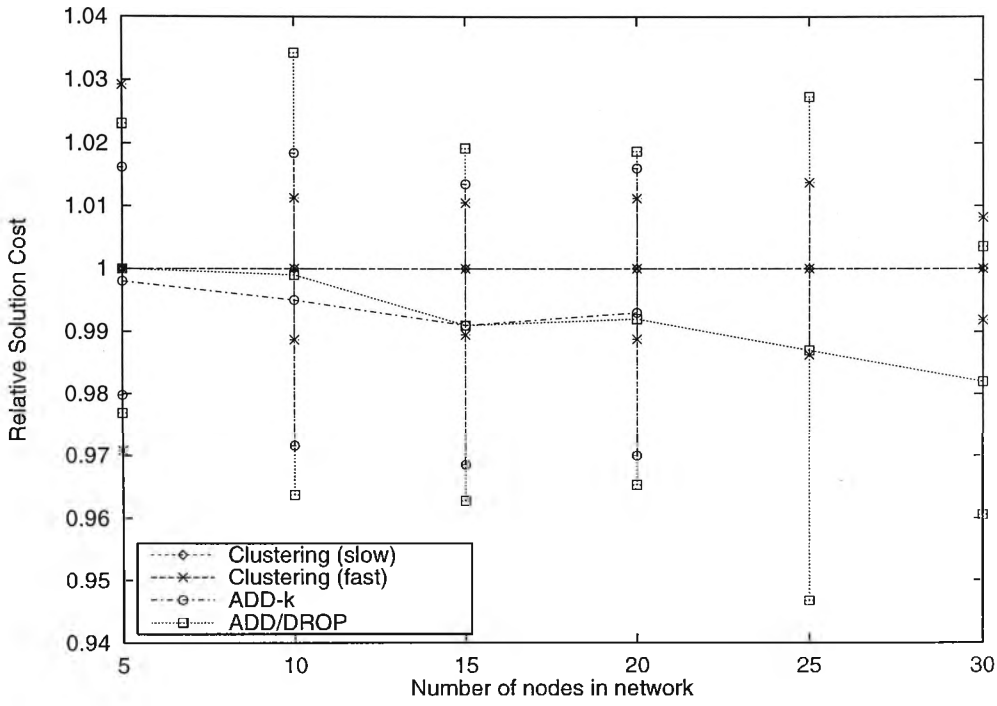


Figure B.1 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-*k* procedures as the size of the network varies. Network topologies are constrained.

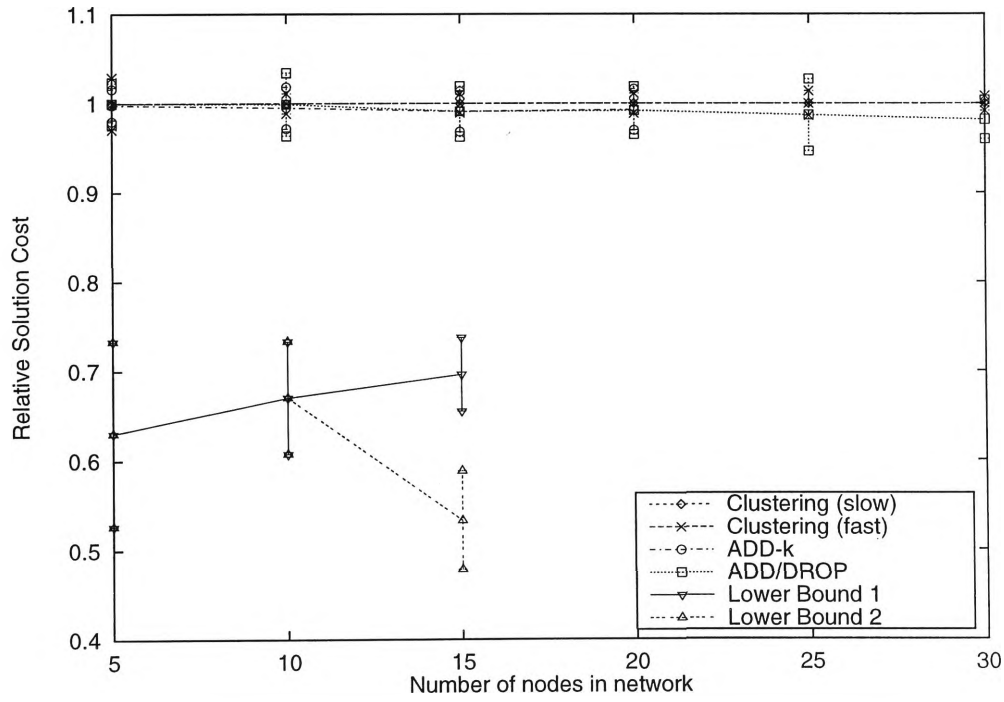


Figure B.2 Relative cost of solutions produced by the clustering, ADD/DROP, ADD-*k*, and lower bounding procedures as the size of the network varies. Network topologies are constrained.

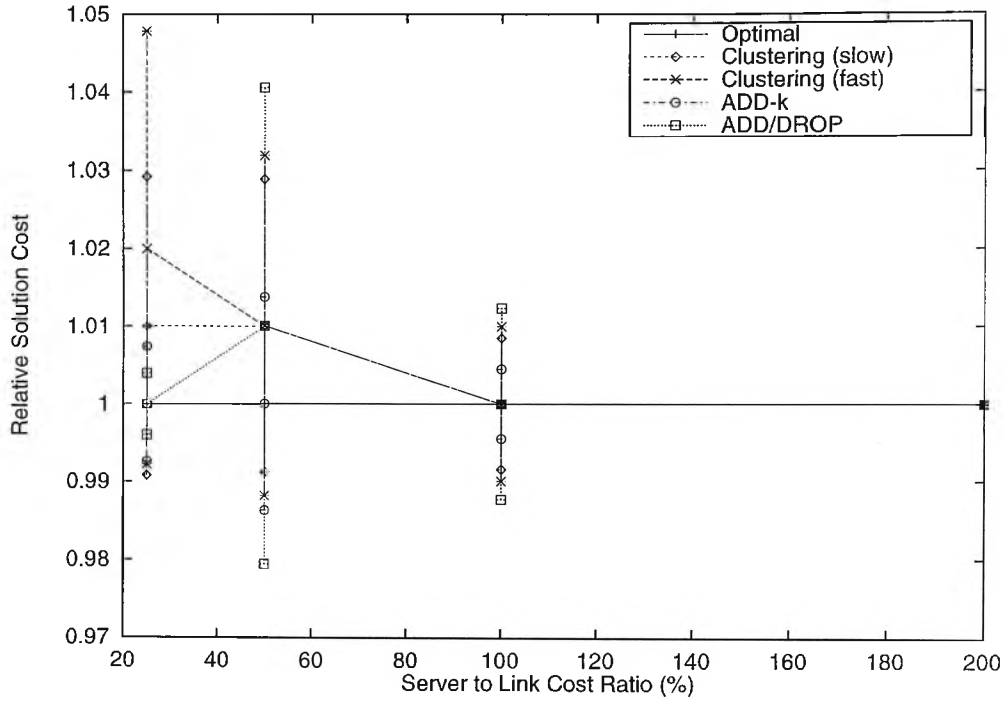


Figure B.3 Relative cost of solutions produced by the clustering, Add/DROP, and ADD-*k* procedures as the relative cost of servers and links varies. Network topologies are constrained.

Figures B.4 and B.5 correspond to Figures 6.12 and 6.13 respectively, the reader is referred to Section 6.4.2.3 for further details.

Figure B.6 corresponds to Figure 6.14, the reader is referred to Section 6.4.2.4 for further details.

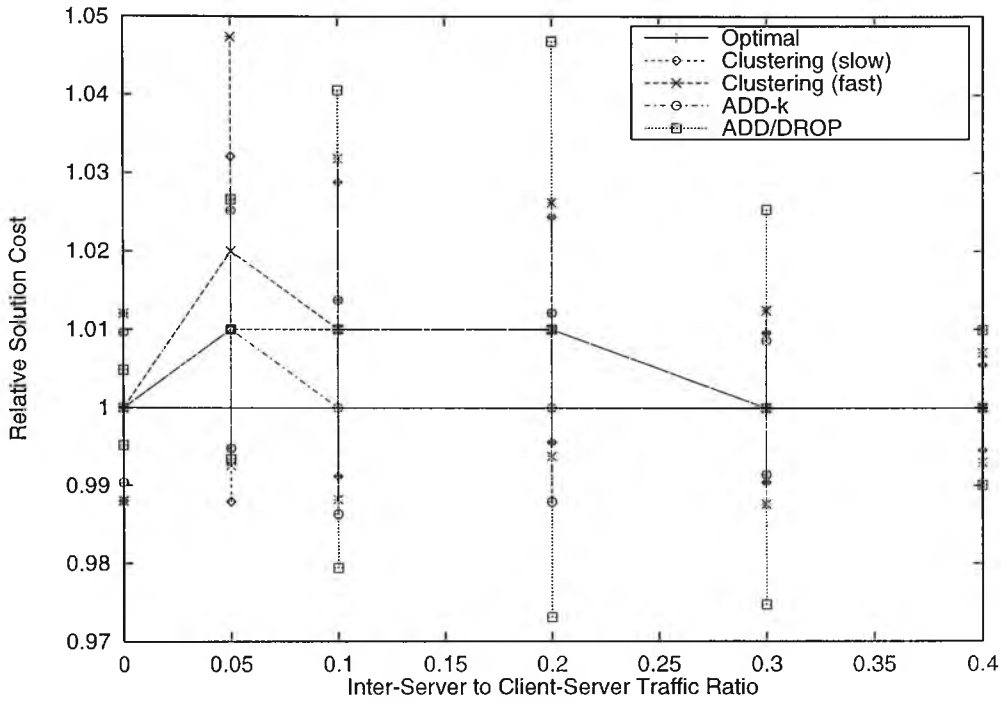


Figure B.4 Relative cost of solutions produced by the clustering, ADD/DROP, and ADD-*k* procedures as the ratio of inter-server to client-server traffic varies. Network topologies are constrained.

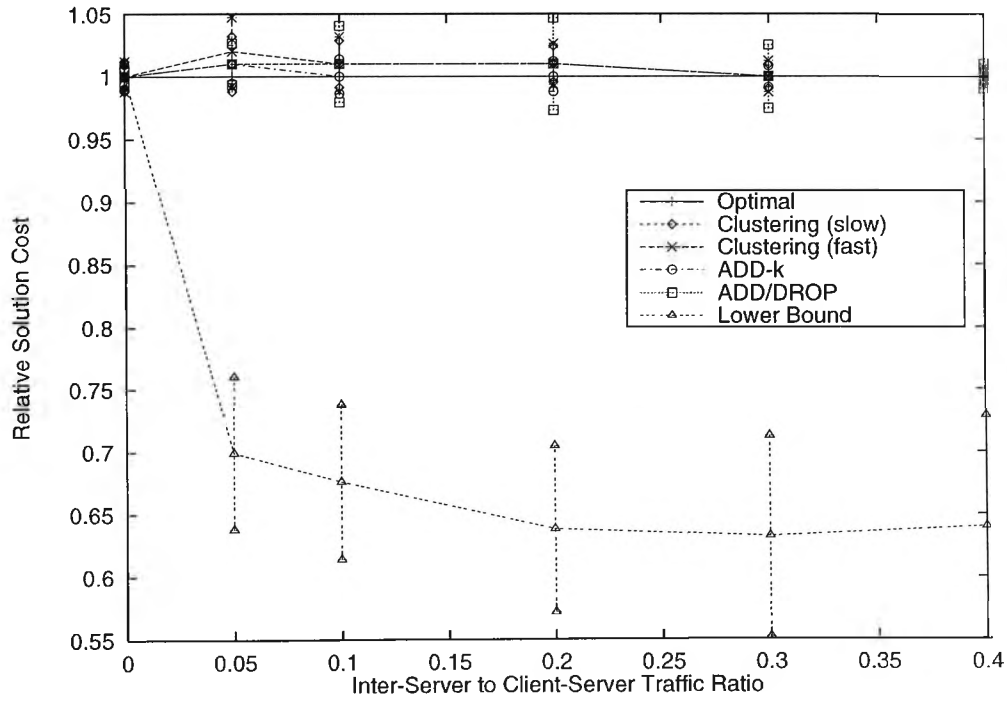


Figure B.5 Relative cost of solutions produced by the clustering, ADD/DROP, ADD-*k*, and lower bounding procedures as the ratio of inter-server to client-server traffic varies. Network topologies are constrained.

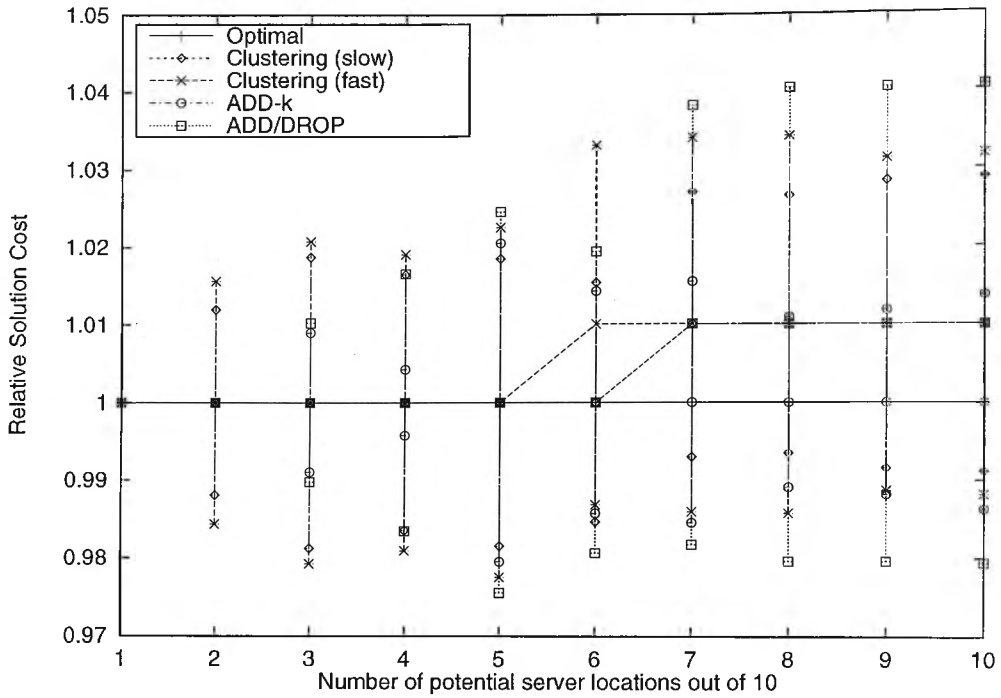


Figure B.6 Relative cost of solutions produced by the clustering, ADD/DROP, and Add-*k* procedures as the number of potential server locations varies. Network topologies are constrained.